НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
Завідувач кафедри
_____ *Сергій СТІРЕНКО*
«___»_____2020 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення комп'ютерних систем»

спеціальності 121 «Інженерія програмного забезпечення»

на тему: «Система допомоги водію в умовах обмеженого простору»

Виконала:
студентка _IV_курсу, групи__ІП-62__
Кібко Олександра Романівна _____
(підпис)

Керівник:
Асистент
Аленін Олег Ігорович _____
(підпис)

Консультант з нормоконтролю:
Професор, доктор технічних наук
Сімоненко Валерій Павлович _____
(підпис)

Рецензент
Доцент, кандидат технічних наук
Коган Алла Вікторівна _____
(підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2020 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення комп'ютерних систем»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ *Сергій СТІРЕНКО*

«___»_____20__ р.

ЗАВДАННЯ

на дипломний проєкт студенту

__Кібко Олександрі Романівні__

1. Тема проєкту _*«Система допомоги водію в умовах обмеженого простору»*_

керівник проєкту Аленін Олег Ігорович, асистент, затверджені наказом по університету від «_*07*_»__*травня*___ 2020 р. №_1081-с_

2. Термін подання студентом проєкту _____

3. Вихідні дані до проєкту _*технічне завдання, теоретичні дані.*_

4. Зміст пояснювальної записки: _*аналіз та огляд існуючих систем, опис математичної задачі та алгоритму роботи, вибір технологій та розробка програмного забезпечення системи, демонстрація роботи та можливостей системи.*_

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормконтроль	Симоненко В.П.		

7. Дата видачі завдання _*01.09.2019 року*_

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	*Затвердження теми роботи*	*1.08.2019*	
2.	*Вивчення та аналіз завдання*	*15.09.2019 – 15.03.2020*	
3.	*Розробка архітектури та загальної структури систем*	*15.03 – 25.03.2020*	
4.	*Розробка структур окремих підсистем*	*25.03.2020 – 05.04.2020*	
5.	*Програмна реалізація системи*	*05.04.2020 – 15.04.2020*	
6.	*Оформлення пояснювальної записки*	*15.04.2020 – 20.05.2020*	
7.	*Передзахист*	*26.05.2020*	
8.	*Захист*	*25.06.2020*	

Студент Олександра КІБКО _____

Керівник Олег АЛЕНІН _____

Анотація

В бакалаврській дипломній роботі розроблено систему допомоги водію для руху в обмеженому просторі, який дозволяє водієві слідкувати за переміщенням автомобілю за допомогою графічної інформації у мобільному пристрої.

Система підтримує роботу з декількома варіантами автомобілів, з можливістю розширити список, дінамічною конфігурацією приміщення та зручним інтерфейсом користувача. Система була розроблена за допомогою наборів сенсорів та мікрокомп’ютеру, використовуючи Java та фреймворку Spring для серверної частини і ReactJS для клієнтської частини. Для взаємодії з системою користувач повинен налаштувати параметри сенсорів, та мати пристрій, що з’єднається з приватною мережею.

Annotation

In this work for a Bachelor's Degree it was developed a helper-application system for drivers working in confined spaces. This system allows driver to monitor vehicle movement using graphical illustration on mobile device.

System supports multi-car working mode with opportunity to increase list, dynamic premise configuration and user-friendly interface. System was developed using sensor array and microcomputer, using Java and Spring framework for a server side and ReactJS for a client side. For system interaction user should set up sensor parameters and be able to connect to a private network via mobile device.

ВІДОМІСТЬ ДИПЛОМНОГО ПРОЄКТУ

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	ІЛАЦ.467100.001 ВП	Відомість проекту	1	
3	А4	ІЛАЦ.467100.002 ТЗ	Технічне завдання	4	
4	А4	ІЛАЦ.467100.003 ПЗ	Пояснювальна записка	65	
5	А3	ІЛАЦ.467100.004 Д1	Користувацькі можливості системи	1	
6	А3	ІЛАЦ.467100.005 Д2	Функціональна схема	1	
7	А3	ІЛАЦ.467100.006 Д3	Структурна схема	1	
8	А4	ІЛАЦ.467100.007 Д4	Лістінг	10	

					ІЛАЦ.467100.001 ВП		
Зм.	*Арк.*	*№ документу*	*Підпис*	*Дата*			
Розробив		*Кібко О.Р.*			*Система допомоги водію в умовах обмеженого простору Відомість дипломного проекту*	Літ.	Аркуш 1 / Аркушів 1
Перевірив		*Аленін О. І.*					
						НТУУ «КПІ ім. Ігоря Сікорського», ФІОТ, гр. ІП-62	
Н. Контр.		*Симоненко В.П.*					
Затверд.							

ТЕХНІЧНЕ ЗАВДАННЯ

до дипломної роботи
освітньо-кваліфікаційного рівня «бакалавр»

на тему: “Система допомоги водію в умовах обмеженого простору”

Київ – 2020 року

ЗМІСТ

					ІЛАЦ.467100.002 ТЗ		
Зм.	*Арк.*	*№ документу*	*Підпис*	*Дата*			
Розробив		*Кібко О.Р.*			*Система допомоги водію в умовах обмеженого простору Технічне завдання*	Літ.	Аркуш 1 / Аркушів 4
Перевірив		*Аленін О. І.*					
Н. Контр.		*Симоненко В.П.*				*НТУУ «КПІ ім. Ігоря Сікорського», ФІОТ, гр. ІП-62*	
Затверд.							

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання поширюється на розробку курсу «Веб-програмування». Область застосування: практичне використання людьми в повсякденнному житті для планування задач.

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання роботи кваліфікаційно-освітнього рівня «бакалавр комп'ютерної інженерії», затверджене кафедрою обчислювальної техніки Національного технічного Університету України «Київський Політехнічний інститут ім. Ігоря Сікорського».

3. МЕТА І ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою даного проекту є розробка системи сенсорів та клієнт-серверного додатку для контролювання положення автомобілю в обмеженому просторі

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки є науково-технічна література з теорії і практики програмування, публікації та наукові статті в Інтернеті з даних питань.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розробляємого продукту

- Простий і інтуїтивно-зрозумілий інтерфейс системи.
- Можливість динамічні конфігурації апаратної частини системи.
- Незалежність – система повинна мати програмну автономність і не залежати від встановленого програмного забезпечення.
- Швидка обробка даних з мінімальними похибками

5.2. Вимоги до програмного забезпечення

- Операційна система MS Windows 98 та новіші версії, Linux-дистрибутиви.
- Доступ до мережі Інтернет.
- Пошуковий браузер.

5.3. Вимоги до апаратної частини

- Комп’ютер на базі процесора Intel або AMD мінімальної потужності.
- Оперативної пам’яті не менше 512 Мбайт.
- Наявність набору сенсорів та мікрокомп’ютеру, налаштованих згідно вимогам

6. ЕТАПИ РОЗРОБКИ

	Дата
Вивчення літератури	10.03.2020
Складання і узгодження технічного завдання	01.04.2020
Створення модулів системи, що розробляється	10.04.2020
Тестування окремих модулів системи	20.04.2020
Допрацювання, налагодження і виправлення помилок	04.05.2020
Оформлення документації дипломної роботи	17.05.2020

ПОЯСНЮВАЛЬНА ЗАПИСКА
до дипломного проєкту
на тему: «Система допомоги водію в умовах обмеженого простору»

Київ – 2020 року

ЗМІСТ

ВСТУП

Цього дня в Україні більше чверті населення мають особисту автівку. За останні роки цей показник стрімко збільшувався, та враховуючи доступність європейського ринку купівлі авто, з великою вірогідністю, процент продовже зростати. Багато автомобілів розміщуються в особистих гаражах, які часто не дуже перевищують габарити самого транспортного засобу. Безумовно, якщо людина часто виявляється в подібній ситуації, у неї виробляється навичка і звичка. Проте, якщо змінити умови, наприклад, в одне і те саме приміщення, яке є авто-майстернею, необхідно розміщувати різні транспортні засоби, завдання ускладнюється. А за несприятливими збігами обставин невдалі дії можуть призвести до пошкодження автомобіля, і як наслідок, позапланованими фінансовими витратами. Схожі ситуації можуть виникнути на закритих автомийках або паркінгах, де обмежено простір для маневрів. В таких умовах навіть досвідченому водієві, який прекрасно почуває габарити свого автомобілю, може знадобитися допомога при переміщенні.

Часто власники гаражів або закритих майданчиків винаходять та організовують різні пристосування з метою спрощення орієнтації під час маневрів. Найбільш популярними серед рішень є намальовані контрастні вказівні лінії розмітки, або встановленні дзеркала, що розміщуються в місцях, придатних для розширення кута огляду при маневрах. Такі варіанти рішень дешеві та доступні в реалізації, але все одно не дають точної інформації про безпеку або небезпеку виконуємої дії, адже лише розширюють поле зору водія. Так само де-хто користуються варіантом, пов’язаним зі встановленням лазерних покажчиків у вигляді ліній, орієнтуючись на які, водій встановлює автомобіль, щоб лінії збігалися з умовними мітками простору. Стежачи за цими збігами при русі, можна бути впевненим в правильному і безпечному розміщенні автомобіля в замкнутому просторі.

В наші часи діджіталізації, більш сучасним підходом вважаються рішення, пов’язані з роботою електронних приладів, точність і швидкість роботи яких мають більш високі характеристики, ніж у середньостатистичної людини-водія. А з огляду на те, що майже 100% водіїв мають електронно обчислювальних апарат у вигляді смартфона або планшета, цілком логічним і правильним буде реалізація рішення, де використовується персональний пристрій, до якого людина вже звикла і однозначно знає як з ним працювати. Сучасні реалії та статистика показують, що частіше людина більш довіряє інформації, отриманої з телефону, наприклад, маршрут на картах, аніж орієнтуючись по предметам з реального життя – табличок з адресою на розі будинку.

Таким чином, і на хвилі сучасних тенденцій, бажання швидко і легко отримувати інформацію і її контролювати, навіть у процесі переміщення автомобілю у замкнутому просторі, виникає необхідність в перегляді підходу до отримання інформації. Можна сказати приведення процесу в звичне для людини русло, яке буде наближене до інших, аналогічних процесів сьогодення. І рішення даної проблеми може бути реалізовано за допомогою звичних для всіх і широко розповсюджених цифрових технологій.

РОЗДІЛ 1
АНАЛІЗ ІСНУЮЧИХ СИСТЕМ

1.1. Опис завдання

Основне завдання даної роботи полягає в проектуванні і створенні зв'язної системи з датчиків, системи отримання та обробки даних і веб-інтерфейсу, яка в онлайн-режимі буде відображати графічне представлення поточного стану автомобіля в заздалегідь заданому обмеженому просторі. Це дозволить користувачеві візуально контролювати процес за допомогою будь-якого пристрою, що знаходиться всередині автомобіля, здатного приймати цю інформацію і візуалізувати її. Слід враховувати, що вхідні умови роботи системи, а саме: габарити використовуваного автомобіля, параметри розміщення датчиків і розміри простору можуть відрізнятися, з чого виникає необхідність динамічної їх зміни. Після установки всіх необхідних конфігурацій, користувач повинен мати можливість слідкувати за зміной цих даних у різних способах їх відображення: графічному і цифровому.

Фізична складова системи містить набір датчиків, що визначають відстань до найближчої перешкоди. Отримані дані відправляються на сервер обробки, який з вхідних параметрів формує кінцевий результат, що відправляється на клієнтський пристрій.

Програмна частина реалізована на клієнт-серверній архітектурі, де на стороні сервера відбуваються математичні обчислення і підготовка необхідних даних для передачі клієнту. Сторона клієнта, в свою чергу, обробляє отримані дані необхідним, для перетворення в графічне відображення чином.

1.2. Ультразвукові датчики

Одним з основних компонентів фізичної складової розроблюємої системи, є датчики вимірювання відстані до об'єкта. Варіантом таких приладів – є ультразвукові датчики.

Ультразвук – пружні коливання і хвилі, частота яких перевищує $(1{,}5-2)*10^4$ ГЦ [1].

Ультразвуковий датчик – сенсорний пристрій, що перетворює електричний струм у хвилі ультразвуку [2].

Принцип дії таких датчиків схожий на роботу радара, вони вловлюють ціль по відображеному сигналу. Частота звукової хвилі, яку використовують прилади, знаходиться в межах частоти ультразвуку, що забезпечує концентрований напрямок звукової хвилі, оскільки звук з високою частотою менше розсіюється у навколишньому середовищі. Оскільки, швидкість поширення звуку в повітряному середовищі – величина постійна ($v_{зв} = 331 м/с$), датчиком обчислюється відстань до деякого об'єкту (S), що відповідає діапазону часу між виходом сигналу (t_1) і його поверненням (t_2). Відстань до об'єкта приблизно дорівнює половині пройденого шляху звукової хвилі.

$$S = \frac{v_{зв} * (t_1 - t_2)}{2} \tag{1.1}$$

Типовий ультразвуковий датчик відстані складається з двох мембран, одна з яких генерує звук, а інша – реєструє відбите ехо. Звуковий генератор створює маленький, з деяким періодом ультразвуковий імпульс і запускає таймер. Друга мембрана реєструє прибуття відбитого імпульсу і зупиняє таймер.

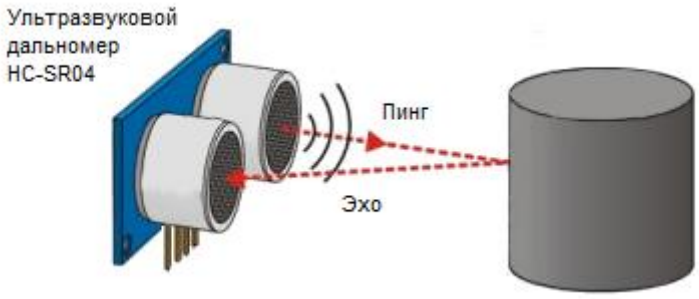


Рис. 1.1 Ілюстрація роботи ультразвукового датчику

Хоча принцип роботи усіх датчиків однаковий, в залежності від використовуваних при його збірці компонент, можуть відрізнятися деякі

параметри вимірювань. Так, для використовуваного датчика в документації вказані такі характеристики:

Таблиця 1.1

Дальність	2 – 500 см
Оглядовий кут	15°
Вимірювальний кут	30°
Сенсорна здатність	0,3 см

Датчик не надає можливості вимірювати великі відстані, але в умовах розроблюємої системи пріоритет віддається точності вимірювань в невеликих масштабах. Таким чином, дальність до 5 метрів є прийнятним значенням, враховуючи стабільність роботи і точність проведених вимірювань.

1.3. Wi-Fi модуль

Ще однією складовою системи, що забезпечує передачу даних між її компонентами є Wi-Fi модуль. Основна функція такого модулю полягає в зборі даних з датчиків, підключення до Wi-Fi точки доступу і передачі даних на сервер обробки.

Найпростішою та найпоширенішою моделлю є ESP-01. Вона має режим роботи як клієнт, так і точка доступу, і може бути використана у якості будь-якого з них, в залежності від написаної програми.

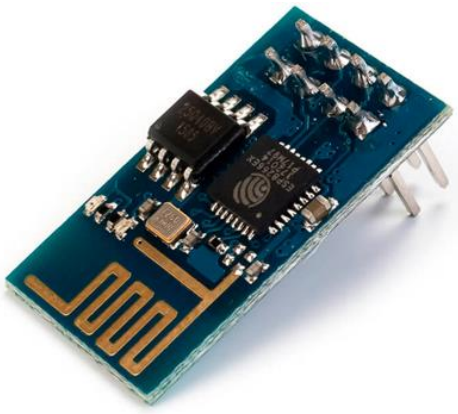


Рис. 1.2 Ілюстрація Wi-Fi модулю ESP-01

Датчик має вбудовану пам’ять в 1 МБ, для запису виконуваного програмного коду. Завантаження програми в дану модель можлива тільки за допомогою зовнішнього контролера, але по закінченню завантаження необхідності в контролері немає. Програма виконується автоматично в циклічному режимі, поки на модуль подається живлення.

1.4. Сервер обробки

Після розміщення необхідної кількості датчиків з’являється необхідність в пристрої збору даних, їх обробки і перетворення. Виходячи з поточних вимог системи, необхідно щоб апаратний пристрій відповідав критеріям:

- Портативність (розмір) пристрій повинен відповідати прийнятним величинам для легкого переміщення і розміщення в будь-якій точці приміщення;
- Обчислювальна потужність повинна задовольняти потребу в швидкому перетворенні отриманих даних з найменшими затримками від необхідної кількості датчиків;
- Метод передачі та отримання даних повинен бути або провідним, якщо це стаціонарне рішення і є така можливість, або краще безпровідним для універсалізації системи.

Виходячи з критеріїв до пристрою одним з придатних варіантів є одноплатний мікрокомп’ютер моделі «Raspberry Pi».

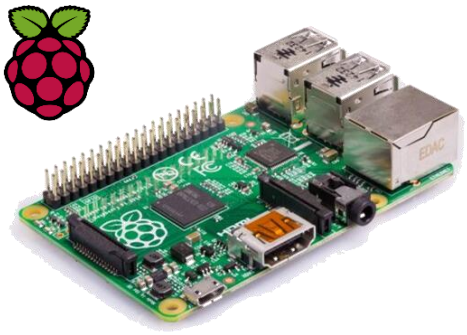


Рис. 1.3 Ілюстрація та логотип мікрокомпєютеру «Raspberry Pi»

Велика частина моделей одноплатних комп'ютерів «Raspberry Pi» поширюється повністю зібраним продуктом на чотирьошаровій платі, розміром не більше за банківську карту.

Стандартна комплектація такого мікрокомп'ютера складається з чіпу Broadcom BCM2835, що включає в себе процесор ARM із тактовою частотою 700 МГц, графічний процесор VideoCore IV, та 512 або 256 мегабайтів оперативної пам'яті. При необхідності зберігання даних, використовуються знімні SD-карти, замість твердих дискових накопичувачів.

Починаючи з 2013 року до сьогодення було випущено 9 версій даного мікрокомп'ютера. З розвитком технологій кожна з них удосконалювала і перевищувала за технічними характеристиками свого попередника.

Таблиця 1.2

№	Процесор	Частота	Ядер	ОЗП	Ethernet,WiFi	Bluetooth
A	ARM1176JZ-F	700 МГц	1	256 МБ	-	-
A+	ARM1176JZ-F	700 МГц	1	256 МБ	-	-
B	ARM1176JZ-F	700 МГц	1	512 МБ	+,-	-
B+	ARM1176JZ-F	700 МГц	1	512 МБ	+,-	-
2B	ARM Cortex-A7	900 МГц	4	1 ГБ	+,-	-
3B	ARM Cortex-A53 x64	1,2 ГГц	4	1 ГБ	+, 802.11n Wi-Fi	4.1
3B+	ARM Cortex-A53 x64	1,4 ГГц	4	1 ГБ	+,2.4/5 ГГц 802.11ac Wi-Fi	4.2
4B	ARM Cortex-A72 x64	1,5 Ггц	4	1-4 ГБ	+, 2,4 ГГц та 5,0 ГГц IEEE 802.11ac	5.0

Так, в перших версіях, працюючи на встановленому процесорі з частотою 700 МГц, «Raspberry Pi» забезпечив продуктивність, приблизно еквівалентну 0,041 GFLOPS. Встановлений графічний процесор здатен працювати з частотою в 1 - 1.5 Gpixel/s при обробці графіки або, якщо

говорити про загально-обчислювальну продуктивність – то значення приблизно складає 24 GFLOPS.

Наступна за датою виходу модель «Raspberry Pi 2» першої версії мала у комплектації 1 Гб оперативної пам'яті та чотирьохядерний процесор Cortex A7, який працює на частоті 900 МГц. Графічний процесор майже не відрізняється від оригінального з першої версії. На відміну від центрального, для якого була зазначена потужність у 4 – 6 разів. Таким чином у паралельних тестах, «Raspberry Pi» другої версії продемонстрував загальну швидкість більшу ніж у 5 – 7 разів, ніж «Raspberry Pi» першої версії моделі B +.

Третя версія «Raspberry Pi», яка складається з чотирьохядерного процесора ARM CortexA53, характеризується десятикратною продуктивністю на відміну від своєї першої моделі. Тестові дослідження довели, що «Raspberry Pi 3» майже на 80% швидше, аніж попередник «Raspberry Pi 2» при виконанні однакових паралельних задач.

Остання модель на сьогодення – «Raspberry Pi 4B», презентована виробниками 23 червня 2019 року. Хоча зовні габарити комп'ютеру збереглися від попередньої серії, а саме – 85,6 мм × 56,5 мм, сама система отримала багато нових покращень та вдосконалень. Нова версія побудована на основі чотирьохядерного процесору 1.5 GHz ARM CortexA72 з частотою 1,5 ГГц. Вбудована Wi-Fi підтримка сберігла рівень стандарту 802.11ac, але підтримка Bluetooth змінилася з версії версії 4.2 до останньої та найпотужнішої 5.0. Істотним кроком вгору стало оновлення у роботі мережевого адаптеру Gigabit Ethernet, який став працювати, використовуючи повну пропускну здатність, не обмежуючи 300 Мбіт/с шлюзу, як у третій серії мікрокомп'ютеру.

Таким чином, використовуючи даний мікрокомп'ютер забезпечується відмінне співвідношення розмір до продуктивності, оскільки, маючи габарити, що не сильно перевищують банківську картку, даний комп'ютер вважається повноцінним і здатний у багатьох потребах замінити звичайний

сервер. Також, аналіз швидко зростаючого ринку даної моделі відображає наявність версій з технологіями Ethernet і Wi-Fi, які повністю покривають необхідність у бездротовій передачі даних.

1.5. Клієнт-серверна архітектура

У системі, легко виділяються дві діючі сторони: система датчиків з сервером обробки даних та кінцевий пристрій користувача, що відображає результат роботи.

Термін "клієнт-сервер" означає таку архітектуру програмного комплексу, у якій функціональні частини взаємодіють по схемі "запит-відповідь". Якщо розглянути дві взаємодіючі частини цього комплексу, то одна з них – клієнт – виконує активну функцію, тобто ініціює запити, а інша – сервер – пасивно на них відповідає.

Застосовуючи таку систему до вимог розроблюємого продукту, чітко видно, що саме пристрій водія виступає в ролі ініціатора запитів – сторона клієнта. Такий пристрій за запитом користувача, наприклад, водій відкрив необхідну веб-сторінку, подає сигнал на сторону сервера, запитуючи дані, необхідні для відображення очікуваного результату. Стороною сервера в даному випадку буде виступати мікрокомп’ютер, що займається обробкою даних з датчиків. Саме на нього надходять запити від клієнта, які активують процес обчислень і підготовки результату до відправки у відповідь на пристрій клієнту.

Часто в описі «клієнт-серверної» архітектури виділяють третій модуль – модуль зберігання даних. Саме ці дані належним чином обробляються стороною сервера і очікуються для відображення на стороні клієнту. У поточному випадку, в ролі даних виступають саме показники датчиків, які є динамічними і унікальними для кожного із запитів. Таким чином, сукупність вимірюваних даних з датчиків і формує модуль даних.

Рис. 1.4 Схема взаємодії модулів клієнт-серверної архітектури

Взаємодія між стороною клієнта і сервером, розташованими на різних фізичних пристроях відбувається за допомогою мережевого з'єднання (дротового або бездротового), використовуючи мережеві протоколи. Так повідомлення з боку клієнту до сервера називають «запитом», а зворотний зв'язок від сервера до клієнту – «відповіддю». Подібне мережева взаємодія описана за допомогою OSI моделі, що складається з 7 рівнів, кожен з яких містить протоколи комунікацій:

- Прикладний рівень – останній рівень моделі, що безпосередньо контролює взаємодію мережі та її користувача. Завдяки цьому рівню додатки на боці користувача мають доступ до мережі і наданих нею служб, наприклад доступ до даних та файлів.
- Рівень представлення – цей рівень відповідає за перетворення протоколів і кодування/декодування даних. Запити додатків, отримані з прикладного рівня, він перетворить у формат для передачі по мережі, а отримані з мережі дані перетворить у формат, зрозумілий додаткам;
- Сеансовий рівень – рівень, що контролює підтримку сеансу зв'язку, між клієнтом та сервером, дозволяючи працюючим додаткам обмін інформацією між собою тривалий час. Рівень безпосередньо контролює створення та завершення сеансу зв'язку, обмін інформацією, визначення дозволу на відправку та отримання даних, підтримку активного з'єднання, коли будь-який додаток знаходиться у режимі неактивності;

- Транспортний рівень – 4-й рівень, що призначено для безпомилкового доставлення даних саме у в тій послідовності, у якій вони були надіслані, контролювання витрат і дублювання;
- Мережевий рівень – 3-й рівень моделі, призначений для контролювання мережевого маршруту відправлення пакетів даних, а саме – визначення шляху їх передачі. Цей рівень також перетворює логічні імена та адреси призначення у їх фізичні відповідності, шукає оптимальні маршрути передачі даних, слідкує та уникає заторів та мережеві поломки;
- Канальний рівень – рівень що забезпечує безпомилково (а у разі виникнення помилок – їх виправлення) взаємодію на фізичному рівні;
- Фізичний рівень – найнижчій рівень моделі, завдяки якому відбувається передача даних.

Таким чином, мережева взаємодія між пристроями забезпечується існуючою моделлю коммуніціювання з використанням HTTP протоколу прикладного рівня. Використання саме цього протоколу обумовлюється кінцевою метою сполучення між пристроями, а саме – відображення інформації на клієнтському пристрої у вигляді веб-сторінки.

HTTP (англ. HyperText Transfer Protocol) – протокол прикладного рівня для передачі гіпертексту [4]. В поточний час у якості надсилаємого «гіпертексту» можуть виступати різні дані, наприклад: HTML документ або JSON об’єкт.

Hypertext Markup Language (HTML) – стандартна мова розмітки документа, призначена для відображення у веб-браузері.

JavaScript Object Notation (JSON) – файловий формат, для зберігання і передачі даних, який складається з читабельного тексту і призначений для легкого перетворення в програмні об’єкти і компоненти.

Виходячи з цього, при необхідності в мережевий передачі даних оптимальним варіантом є комунікація з використанням протоколу HTTP,

який в свою чергу, найчастіше використовується для відображення інформації на веб-сторінці.

Використовуючи HTTP протокол, найбільш часто зустрічається інтерфейс спілкування між клієнтом і сервісом під назвою REST (укр. передача стану уявлення). REST, на відміну від протоколів, тільки описує принцип організації «спілкування» сайту та його серверної складової. За засобами даного інтерфейсу взаємодія з серверною частиною зводиться до чотирьох базових операцій, відповідаючих методам HTTP-запиту:

- Отримання даних з серверу – GET метод
- Додавання нових даних на сервер –POST метод
- Зміна існуючих даних на сервері – PUT метод
- Видалення даних з сервера – DELETE метод

Крім використовуваних операцій, концепція REST передбачається передачу інформації про користувача у кожному відправленому на сервер запиті, замість запам’ятовування стану між запитами.

1.6. Технології серверної частини

Для реалізації клієнт-серверної архітектури, програма, що розроблюється, повинна відповідати всім вимогам і бути здатною реалізувати роботу серверної сторони додатку.

Найбільш популярним серед розробників є фрейморк на мові програмування Java – Spring Boot. Він містить усі необхідні компоненти для швидкого розгортання повноцінного додатку. Найважливіші з них для створення веб-додатків – це вбудований контейнер сервлетів (він же веб-сервер) і керування кінцевими точками REST.

Для конфігурації Java-додатку, Spring надає набір анотацій, які згодом будуть автоматично перетворені в xml-конфігурації, необхідні для роботи програми. Серед необхідних аноацій слід виділити наступні:

- @SpringBootApplication – точка входу в додаток, аналог main методу в звичайному Java додатку;

- @Controller – анотація класу, для конфігурації його методів в обробники кінцевих точок REST;
- @Service – анотація класу з бізнес-логікою додатку, для виділення її реалізації від обробника запитів;
- @Component – анотація класу, що реалізує додаткові допоміжні методи, що використовуються у будь-якій частині програми.

Spring Boot підтримує використання Apache Maven, як автоматизації процесу налаштування і збірки додатку. Зазначена конфігурація в файлі pom.xml проекту дозволяє швидко підключити необхідні надбудови і бібліотеки до проекту, так само як і конфігурувати процес його складання в готовий до використання артефакт.

При використанні типу JSON під час передачі необхідних даних в запитах, методи контролер класів Spring додатку автоматично перетворять дані в обидві сторони: з формату JSON в Java об’єкт, поля якого збігаються з описом заначень в JSON. І зворотне перетворення – з Java об’єкта в формат JSON, шляхом формування назви атрибуту з імені поля об’єкту, і подальшого запису його значення.

Таким чином, розробка з використанням фреймворку Spring Boot потребує написання безпосередній логіки і конфігурації, які стосуються тільки бізнес-мети проекту, який реалізовується. Усі додаткові конфігурації, такі як налаштування контейнера сервлетів, і запуску додатка на ньому зводяться до мінімуму.

1.7. Технології клієнтської частини

Можливості сторони клієнту повинні так само відповідати вимогам клієнт-серверної архітектури. Серед них: можливість взаємодії з стороною серверу, засобом відправки HTTP запитів, реалізація веб-інтерфейсу з усіма необхідними для його використання компонентами.

Одним з варіантів технології для реалізації поставленої задачі є фреймворк ReactJS, для розробки і відтворення интерфейсу користувача.

Фреймворк взаємодіє з бібліотеками Redux і Axios, які окремо підключаються до проекту, для контролювання запитів на сторону серверу.

Використання ReactJS дозволяє реалізувати елементи інтерфейсу в окремих компонентах, на подобі класів у Java. Після визначення вмісту компоненти у файлі, з'являється можливість її використання з іншої компоненти, у вигляді тега, замість якого при відображенні на HTML-сторінці додається код, розміщений у файлі компоненти. Такий зв'язок між ними реалізує батьківсько-дочірню модель, де в ролі дочірнього елемента виступає кінцева компонента, а у ролі батька – компонента, що її використовує. ReactJS підтримує передачу властивостей між компонентами. При цьому такий зв'язок є одностороннім і переходить від батька до дочірнього елементу, що дозволяє контролювати весь набір даних в одному місці, а передавати необхідні частини для відображення – в іншому.

Кожна компонента має методи життєвого циклу, аналогічні методам Java об'єкту. Описані у них дії виконуються на певних етапах обробки.

- Constructor – конструктор компоненти, який викликається перед її відтворенням, встановлює необхідні для роботи значення змінних і констант;
- ComponentDidMount – викликається після первинного відтворення компоненти, в основному використовується для відправки запиту на отримання даних з боку сервера;
- ComponentWillReceiveProps – викликається за необхідністю підвантаження нових значень, що зберігаються в стані компоненти, після їх отримання;
- Render – основний метод кожної компоненти, без якого не відбувається відтворення. Метод викликається кожен раз при зміні даних всередині компоненти.

Для передачі запитів і обробки відповідей від серверу використовуються сторонні бібліотеки Redux і Axios. З них бібліотека Axios використовується безпосередньо для управліннями запитами і роботою з JSON даними.

Бібліотека підтримує всі види методів HTTP, приймає як параметр кінцеву точку запиту, та за наявністю його тіло. У якості результату повертає відповідь від серверу після її отримання. Бібліотека Redux, в свою чергу, є менеджером стану і дозволяє зберігати всі необхідні дані, в наданому сховищі.

В цілому, розробка інтерфейсу за допомогою ReactJS схожа з верстkою HTML-сторінок, зі вставками java-script. Так при цьому, використовуючи компоненти ReactJS, теги для відтворення елементів, функції, що ними використовуються визначаються в одному місці.

1.8. Аналіз існуючих рішень

У більшості сучасних автомобілів, що обладнані електронікою, присутні різні системи допомоги водієві. Кожна компанія, яка випускає автомобіль з вбудованою системою або ж окремий аналог, який потребує окремої установки, має унікальний дизайн і інтерфейс програми, проте принцип роботи здебільшого залишається незмінним.

Одним з варіантів – є камера заднього ходу. Такий пристрій встановлюється на задню частину автомобіля, та забезпечує комфортне управління і контроль відстані при русі заднім ходом. У таких пристроях інформація про рух виводиться у візуальному і звуковому вигляді, з графічними позначками які дають інформацію про відстань до видимих об’єктів полі зору камери. Однак, подібні пристрої працюють у вузьконаправленому діапазоні, який обмежено кутом огляду і напрямком камери.

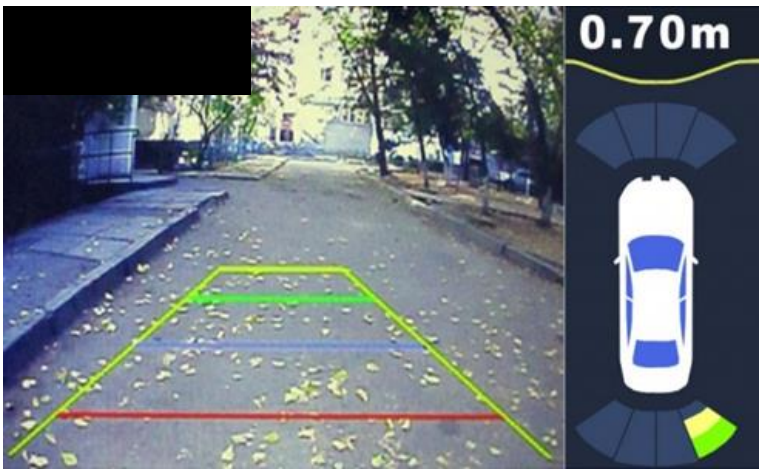


Рис. 1.5 Приклад роботи камери заднього руху

Повноцінний або частковий «парктроник» є найближчим аналогом розроблюваного програмно-апаратного рішення. Такий пристрій являє собою один датчик або їх систему, які отримують інформацію про перешкоди, що оточують автомобіль. У простіших реалізаціях, інформація з датчиків виводиться на дисплей у вигляді цифр і мінімального графічного інтерфейсу зі звуковим інформуванням, щоб водій міг контролювати маневри автомобілем з деякою точністю, яка властива для даної моделі системи.

Рис. 1.6 Приклад роботи цифрового парктронику

Наступним варіантом є повноцінний парктроник, що має доступ до керування автомобілем та за необхідності може замінити водія у процесі паркування. В такій системі датчики розміщуються в декількох місцях автомобіля, що збільшує площу покриття і дозволяє контрольовано здійснювати не тільки їзду заднім ходом, а й різні маневри, наприклад, паралельну парковку або розворот в обмеженому просторі.

Однак, у використанні парктроника є свої недоліки. У разі, коли рівень розташування датчика перевищує висоту перешкоди (наприклад низька бровка), яка в такому випадку залишиться непоміченою без людського втручання і може призвести до пошкодження машини. Або ж зворотний випадок, коли датчик буде реагувати на несуттєві деталі, наприклад, висока трава, та тим самим дезорієнтує водія, не залишаючи однозначної відповіді чи є перешкода на шляху – чи ні.

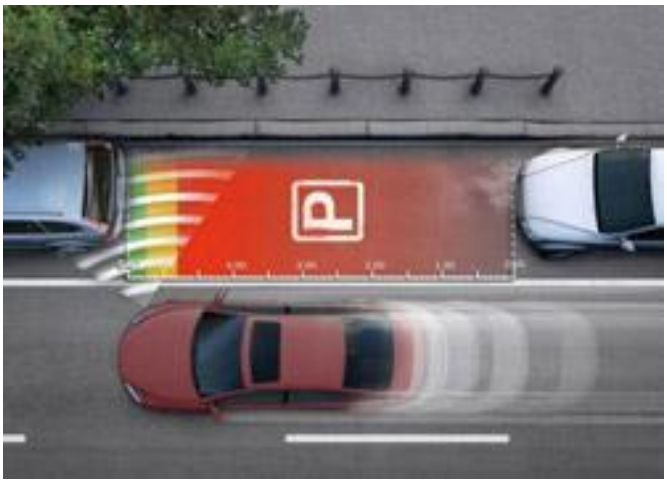


Рис. 1.7 Схема роботи системи «Парктроник»

За останніми дослідженнями, середній вік українського парку автомобілів складає 12-15 років. Такі автомобілі, випущені з мінімальною комплектацією, в своїй більшості не мають вбудованих систем допомоги водієві. Звичайно, такі удосконалення легко придбати і можуть бути «додані» до автомобіля, як звичайна система сигналізації або магнітола. Однак варто пам’ятати, що велика кількість сучасної електроніки вимагає живлення, яке надається машині акумулятором і/або генератором. Таким чином, зі збільшенням кількості приладів в автомобілі, збільшується споживання ними електрики, і підвищене навантаження на генератор, що негативно впливає на робочу функціональність акумулятору, а саме – збільшує ймовірність прискореного розряду або хронічного недозаряду, що безпосередньо сприяє виходу з ладу акумулятора, і як наслідок – неможливості завести автомобіль. З огляду на таку специфіку, зовнішні прилади для допомоги при парковці, стають прийнятним компромісним рішенням.

ВИСНОВКИ ДО РОЗДІЛУ 1

У розділі було наведено опис фізичної складової рішення, разом з описом роботи складових результуючої системи. Детально розглянуто фізику процесу отримання необхідних для обробки даних, та обґрунтовано потреби та оптимальний вибір використаного обладнання. Також було розглянуто програмну складову, описано обрану модель опрацювання та передачі даних між пристроями, що приймають участь у роботі системи.

Також, у розділі було наведено опис проблеми, з якою стикаються багато водіїв, як з досвідом керування автівкою, так і початківці. Після аналізу існуючих рішень та можливості їх застосування в умовах нашої країни, була пояснена актуальність поставленої задачі. Розглянуті рішення мали свої переваги і недоліки, та попри це залишаються випадки, коли існуючі технології не можуть допомогти у виникаючих обставинах.

РОЗДІЛ 2
ОПИС МАТЕМАТИЧНОЇ ЗАДАЧІ

2.1. Загальна геометрична модель

Основним завданням поточної системи є найбільш точне визначення положення автомобіля при мінімальній кількості вхідних параметрів. Вхідні параметри діляться на постійні і змінні. Постійні представляють собою статичні значення, рівні габаритам автомобіля, що використовується та оточуючого приміщення. Динамічні – це значення, які визначаються датчиками в поточний момент часу.

Розроблена геометрична модель являє собою чотири датчика, розміщених в найбільш вдалих для розрахунків місцях. Таким чином, для коректної її роботи необхідні наступні положення:

- Датчик A (передній) – розміщується на дальній стінці приміщення, максимально наближено до середини по ширині під кутом 90 градусів до самої стіни: $A \in NO, NA \cong AO$;
- Датчик B (лівий кутовий) – розміщується на лівій стінці, на відстані більшій або рівній половині довжини бокової стіни, навпроти датчика C, під кутом в 30 градусів: $B \in NM, NB \geq \frac{NM}{2}, NB = OC$;
- Датчик C (правий кутовий) – розміщується на правій стінці, на відстані не менше половини довжини бокової стіни, навпроти датчика B, під кутом в 30 градусів: $C \in OP, OC \geq \frac{OP}{2}, NB = OC$;
- Датчик D (лівий бічний) – розміщується на лівій стінці, максимально близько до в'їзду в приміщення (ближче ніж лівий кутовий), під кутом в 90 градусів: $D \in MB$.

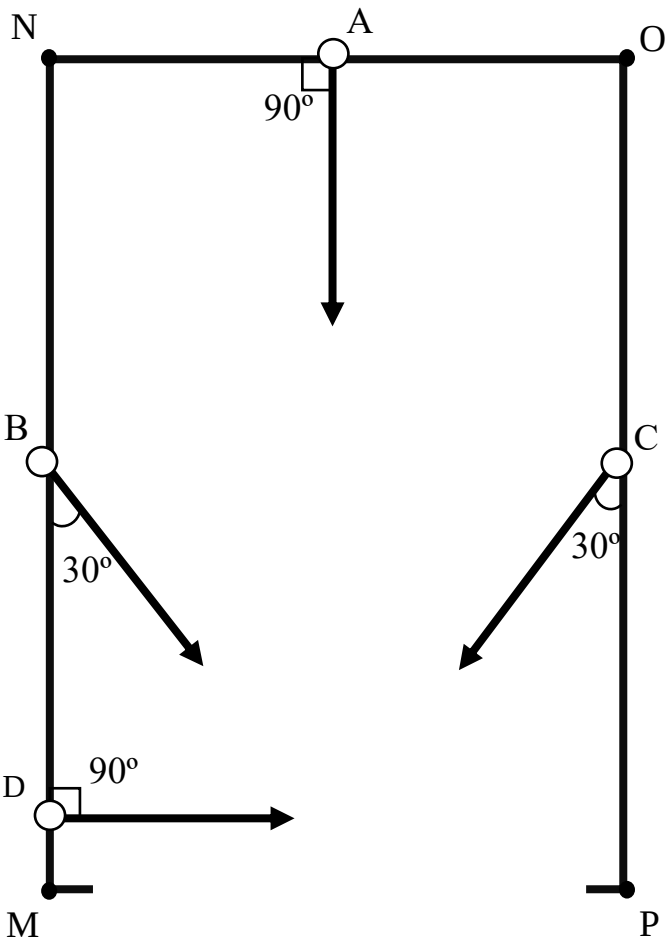


Рис. 2.1 Схема розміщення датчиків у приміщенні

Для однозначного визначення положення автомобілю необхідно розрахувати кут повороту, під яким він розташований щодо приміщення, разом зі значеннями дистанції від лівої та передньої стіни.

2.2. Визначення куту повороту

Для визначення куту повороту використовуються кутові датчики B та C. Завдяки їх розташуванню і напрямку дії, отриманні дані дозволяють з найбільшою точністю визначити поточний кут розвороту автомобіля.

Для розробки математичного алгоритму, необхідно розмістити умовну координатну вісь так щоб вісь Ox відповідала стіні з в'їздом, а вісь Oy збігалася з центром приміщення по ширині.

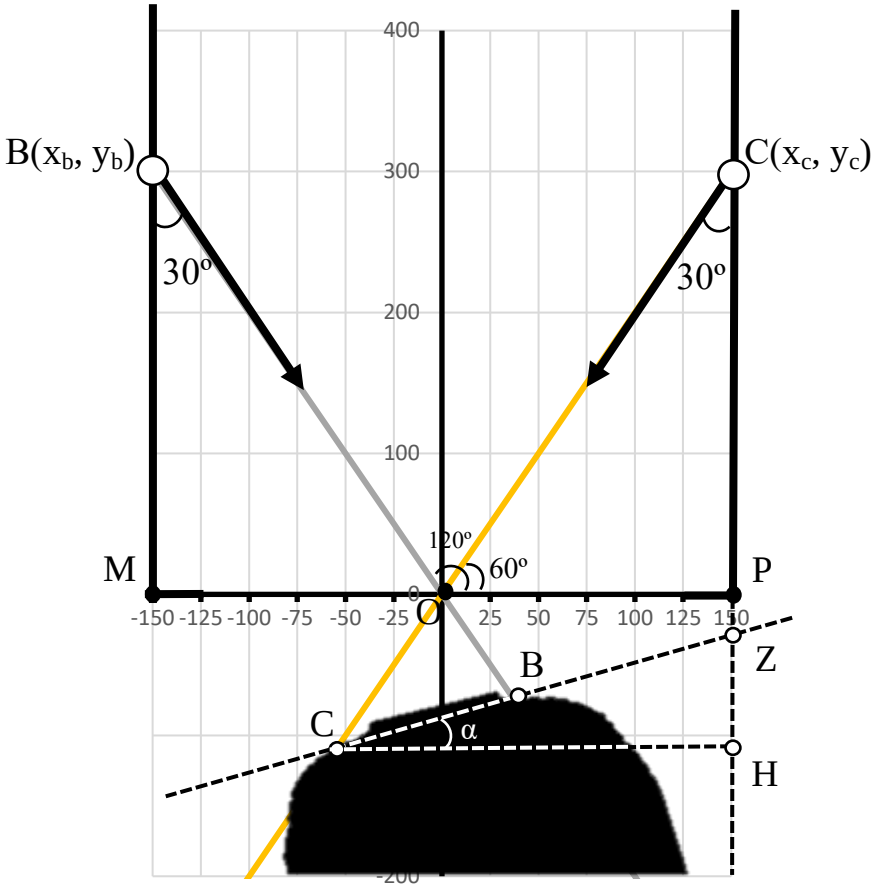


Рис. 2.2 Геометрична модель визначення куту

Для визначення кута повороту маємо наступний набір вхідних даних:

- Координати точки B: x – зворотнє значення половини ширини приміщення $x_b = -\frac{MP}{2}$, y – різниця довжини приміщення і відстані розміщення датчика $y_b = NM - NB$;
- Координати точки C: x – значення половини ширини приміщення ($x_c = \frac{MP}{2}$), y – різниця довжини приміщення і відстані розміщення датчика $y_c = OP - OC$;
- Довжина відрізка BB`, яка рівна показаннями датчика B;
- Довжина відрізка CC`, яка рівна показаннями датчика C;
- Кут нахилу напрямку датчика B до осі Ox: $\angle BOP = 120^\circ$;
- Кут нахилу напрямку датчика C до осі Ox: $\angle COP = 60^\circ$;

В першу чергу необхідно знайти рівняння прямих BB` і CC` у вигляді $y = kx + b$. Де k – кутовий коефіцієнт, що дорівнює тангенсу кута нахилу

прямої до осі Ox. b – зсув графіка функції вздовж осі Oy, яке можна виразити з рівняння прямої з заданим кутовим коефіцієнтом, що проходить через точку $M(x_1, y_1)$: $(y - y_1) = k * (x - x_1) \Leftrightarrow y = kx + (y_1 - kx_1)$. (2.1)

Таблиця 2.1

	k	$b = (y_1 - kx_1)$	$y = kx + b$
BB`	$\tan 120^\circ = -\sqrt{3}$	$y_b + \sqrt{3}\, x_b$	$y = -\sqrt{3}x + y_b + \sqrt{3}\, x_b$
CC`	$\tan 60^\circ = \sqrt{3}$	$y_c - \sqrt{3}\, x_c$	$y = \sqrt{3}x + y_c - \sqrt{3}\, x_c$

Маючи рівняння прямих, координати точок B та C, а також довжини відрізків BB` і CC`, необхідно знайти координати точок B` і C`. Для цього складемо систему зі знайденого раніше рівняння прямої, і також формули довжини відрізка за двома заданими точками $M_1(x_1, y_1)$и $M_2(x_2, y_2)$: $d^2 = (x_2 - x_1)^2 + (y_2 - y_1)^2$. (2.2)

Для пошуку координати точки B` система буде мати вигляд: (2.3)

$$\begin{cases} y = -\sqrt{3}x + y_b + \sqrt{3}\, x_b \\ BB'^2 = (x - x_b)^2 + (y - y_b)^2 \end{cases} \Leftrightarrow$$

$$\Leftrightarrow \begin{cases} y = -\sqrt{3}x + y_b + \sqrt{3}\, x_b \\ BB'^2 = (x - x_b)^2 + \left(-\sqrt{3}x + y_b + \sqrt{3}\, x_b - y_b\right)^2 \end{cases} \Leftrightarrow$$

$$\Leftrightarrow \begin{cases} y = -\sqrt{3}x + y_b + \sqrt{3}\, x_b \\ BB'^2 = (x - x_b)^2 + 3 * (x_b - x)^2 \end{cases} \Leftrightarrow$$

$$\Leftrightarrow \begin{cases} y = -\sqrt{3}x + y_b + \sqrt{3}\, x_b \\ 4x^2 - 8x_b x + 4{x_b}^2 - BB'^2 = 0 \end{cases} \Bigg| \begin{array}{c} D = (-8x_b)^2 - 4 * 4 * (4{x_b}^2 - BB'^2) = \\ = 16 * BB'^2 \end{array}$$

$$\Leftrightarrow \begin{cases} y = -\sqrt{3}x + y_b + \sqrt{3}\, x_b \\ x = x_b \pm \dfrac{BB'}{2} \end{cases} \Leftrightarrow \left[\begin{array}{l} \begin{cases} x = x_b + \dfrac{BB'}{2} \\ y = \dfrac{-\sqrt{3} * BB'}{2} + y_b \end{cases} \\ \begin{cases} x = x_b - \dfrac{BB'}{2} \\ y = \dfrac{\sqrt{3} * BB'}{2} + y_b \end{cases} \end{array} \right.$$

Оскільки шукана точка B` розташована на прямій нижче B, з двох одержаних рішень системи необхідно вибрати значення, де $x > x_b$.

Таким чином, точка B` має координати: $B'\left(x_b+\frac{BB'}{2},\frac{-\sqrt{3}*BB'}{2}+y_b\right)$.

Аналогічний алгоритм для пошуку координат точки C`. Знаючи, довжину відрізка CC` і координати точки $C(x_c,y_c)$ маємо систему: (2.4)

$$\begin{cases} y=\sqrt{3}x+y_c-\sqrt{3}\,x_c \\ CC'^2=(x-x_c)^2+(y-y_c)^2 \end{cases}\Leftrightarrow$$

$$\Leftrightarrow\begin{cases} y=\sqrt{3}x+y_c-\sqrt{3}\,x_c \\ CC'^2=(x-x_c)^2+\left(\sqrt{3}x+y_c-\sqrt{3}\,x_c-y_c\right)^2 \end{cases}\Leftrightarrow$$

$$\Leftrightarrow\begin{cases} y=\sqrt{3}x+y_c-\sqrt{3}\,x_c \\ CC'^2=(x-x_c)^2+3*(x-x_c)^2 \end{cases}\Leftrightarrow$$

$$\Leftrightarrow\begin{cases} y=\sqrt{3}x+y_c-\sqrt{3}\,x_c \\ 4x^2-8x_cx+4{x_c}^2-CC'^2=0 \end{cases}\Bigg|\begin{matrix} D=(-8x_c)^2-4*4*(4{x_c}^2-CC'^2)= \\ =16*CC'^2 \end{matrix}$$

$$\Leftrightarrow\begin{cases} y=\sqrt{3}x+y_c-\sqrt{3}\,x_c \\ x=x_c\pm\frac{CC'}{2} \end{cases}\Leftrightarrow\left[\begin{matrix}\begin{cases} x=x_c+\frac{CC'}{2} \\ y=\frac{\sqrt{3}*CC'}{2}+y_c \end{cases} \\ \begin{cases} x=x_c-\frac{CC'}{2} \\ y=-\frac{\sqrt{3}*CC'}{2}+y_c \end{cases}\end{matrix}\right.$$

Оскільки шукана точка C` розташована на прямій нижче C, з двох одержані рішень системи необхідно вибрати значення, де $x<x_c$. Таким чином, точка C` має координати: $C'\left(x_c-\frac{CC'}{2},-\frac{\sqrt{3}*CC'}{2}+y_c\right)$.

Визначивши координати точок B` і C`, необхідно знайти рівняння прямої, що проходить через ці дві точки. Використовуючи рівняння прямої $\frac{x-x_1}{x_2-x_1}=\frac{y-y_1}{y_2-y_1}$, що проходить через дві задані точки $M_1(x_1,y_1)$и $M_2(x_2,y_2)$:

$$\frac{x-x_{b\prime}}{x_{c\prime}-x_{b\prime}}=\frac{y-y_{b\prime}}{y_{c\prime}-y_{b\prime}}\Leftrightarrow(x-x_{b\prime})*(y_{c\prime}-y_{b\prime})=(y-y_{b\prime})*(x_{c\prime}-x_{b\prime})\Leftrightarrow$$

$$\Leftrightarrow x*(y_{c\prime}-y_{b\prime})-x_{b'}y_{c'}+x_{b'}y_{b'}=y*(x_{c\prime}-x_{b\prime})-x_{c'}y_{b'}+x_{b'}y_{b'}$$

$$\Leftrightarrow y*(x_{c\prime}-x_{b\prime})=x*(y_{c\prime}-y_{b\prime})-x_{b'}y_{c'}+x_{c'}y_{b'}\Leftrightarrow$$

$$\Leftrightarrow y=\frac{y_{c\prime}-y_{b\prime}}{x_{c\prime}-x_{b\prime}}x+\frac{x_{c'}y_{b'}-x_{b'}y_{c'}}{x_{c\prime}-x_{b\prime}}. \quad (2.5)$$

Маємо рівняння прямої B`C`, виражене через координати відомих точок $B'(x_{b\prime}, y_{b\prime})$та $C'(x_{c\prime}, y_{c\prime})$:

$$y = \frac{y_{c'} - y_{b'}}{x_{c'} - x_{b'}} x + \frac{x_{c\prime} y_{b\prime} - x_{b\prime} y_{c\prime}}{x_{c'} - x_{b'}}. \quad (2.6)$$

Для визначення шуканого кута α використовуємо прямокутний трикутник ZC`H, де C`H – перпендикуляр, що опущено з точки C` на пряму CP, Z – точка перетину прямої B`C` з CP. Таким чином, $\cos\alpha = \frac{\text{C`H}}{\text{C`Z}} \Leftrightarrow \alpha = arc\cos(\frac{\text{C`H}}{\text{C`Z}})$.

Для знаходження C`Z визначимо координати точки Z. Оскільки $Z \in CP$, $x_z = x_c$. Знаючи рівняння прямої B`C`, підставимо значення $x = x_z = x_c$, і знайдемо координату y точки Z:

$$y_z = \frac{y_{c'} - y_{b'}}{x_{c'} - x_{b'}} x_c + \frac{x_{c'} y_{b'} - x_{b'} y_{c'}}{x_{c'} - x_{b'}}. \quad (2.7)$$

З формули довжини відрізка між двома відомими точками, знайдемо довжину C`Z, де $\text{C`}(x_{c\prime}, y_{c\prime})$ та $Z(x_z, y_z)$:

$$\text{C`Z} = \sqrt{(x_z - x_{c\prime})^2 + (y_z - y_{c\prime})^2} = \sqrt{(x_\text{c} - x_{c\prime})^2 + (y_z - y_{c\prime})^2} \quad (2.8)$$

Оскільки, відрізок C`H ∥ $Ox$, $H \in CP$ та $x_h = x_c$ його довжина дорівнює $\text{C`H} = x_h - x_{c'} = x_c - x_{c'}$.

Відповідно, значення шуканого кута повороту дорівнює:

$$\alpha = arc\cos(\frac{x_c - x_{c'}}{\sqrt{(x_\text{c} - x_{c'})^2 + (y_z - y_{c'})^2}}) \quad (2.9)$$

Розглянута ситуація та отримана формула розрахунку куту повороту відповідають випадку, коли автомобіль повернено проти годинникової стрілки. Для ситуації, коли поворот проводиться по годинникової стрілки маємо аналогічний малюнок: прямокутний трикутник ZC`H, де C`H – перпендикуляр, опущений з точки C` на пряму CP, Z – точка перетину прямої

B`C` з CP. Таким чином, формула шуканого кута не відрізняється в зазначених двох випадках.

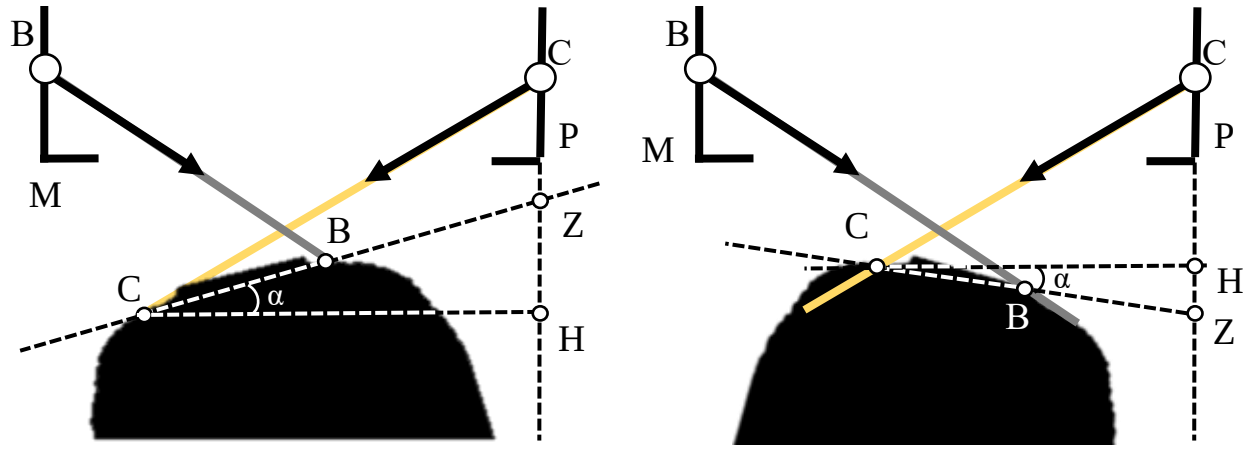


Рис. 2.3 Геометричні моделі випадків повернення у різних напрямах

Однак, для спрощення процесу повернення результату встановимо, що шуканий кут повороту завжди визначається відносно ходу годинникової стрілки. Таким чином, якщо показники датчику B перевищують показники датчика C, що відповідає зоровому повороту проти годинникової стрілки, шуканий кут дорівнює α. А при зворотній ситуації, якщо показники датчику C перевищують показники B – шуканий кут дорівнюватиме $360° - \alpha$.

В результаті маємо:

$$\text{Кут повороту} = \begin{cases} \alpha, \text{при BB`} > \text{CC`} \\ 360° - \alpha, \text{при CC`} > \text{BB`} \end{cases} \quad (2.10)$$

2.3. Визначення положення автомобілю, при відомому куту повороту

Для визначення положення автомобілю використовуються дані з датчиків A та D, визначений раніше кут повороту і зображення автомобіля в пропорційному масштабі. Завдяки розташуванню датчиків і можливостям при роботі з зображенням, визначаються координати розміщення зображення для його пропорційного відображення.

Для розробки алгоритму в першу чергу необхідно зробити поворот зображення на знайдену величину кута. Хоча контур автомобіля має

неправильну форму, будь-яке зображення зберігається на прямокутному полотні, вимірювання якого рівні величинам найширших точок зображення в довжину і ширину. Оскільки поворот проводиться щодо центру зображення, яке для прямокутника є точкою перетину діагоналей, полотно так само змінює свої вимірювання.

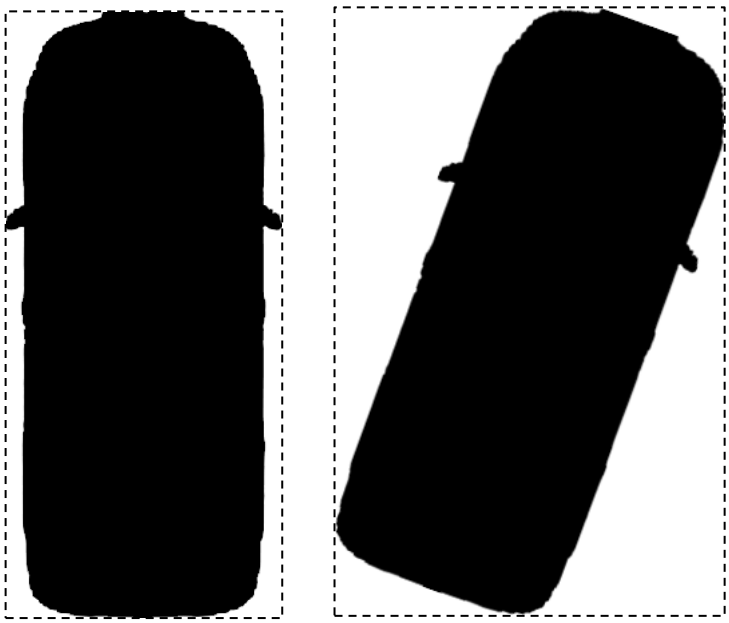


Рис. 2.4 Приклад межі полотна навколо зображення

Для визначення положення автомобілю в першу чергу необхідно зробити поворот зображення на знайдений кут. Оскільки контур, який зображено на малюнках, є пропорційним відображенням реальних габаритів автомобілю, для визначення відстані можна використовувати кількість пікселів білого (у разі відстані до автомобілю) і чорного (габарити автомобіля) кольорів. Після повороту автомобілю з'являється можливість визначити значення відстані від початку полотна до контуру у кожній точці по довжині і висоті зображення.

Розмістимо умовну координатну вісь таким чином, щоб центр збігався з точкою N, вісь Ox – з відрізком NM, Oy – з відрізком NO. Маючи показники датчиків A і D, відповідно рівні AA` і DD`, положення датчиків $A = NA$ і $D = ND$ необхідно знайти координати точки $S(x_s,\ y_s)$, які будуть характеризувати положення зображення.

Слід зазначити, що діапазон припустимих значень координат точки С не повинен перевищувати значення положення датчиків. Таким чином: $S(0 \leq x_s \leq NA, 0 \leq y_s \leq ND)$. А показники датчиків в будь-який момент часу рівні:

$$\begin{cases} \text{AA`} = \text{A}Y + Y\text{A`}, \text{где A}Y = x_s \\ DD` = \text{D}X + XD`, \text{где D}X = y_s \end{cases} \quad (2.11)$$

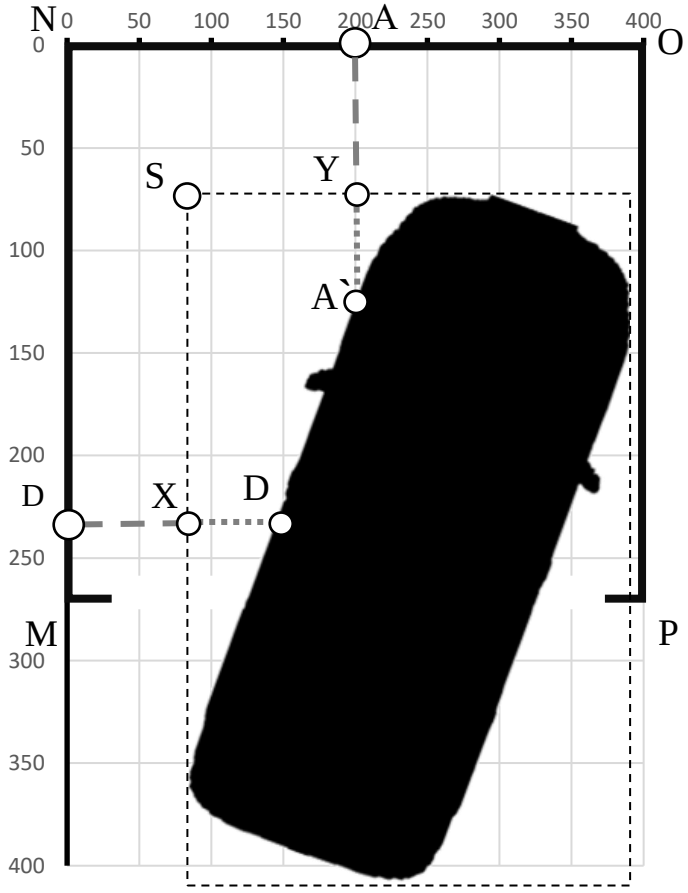


Рис. 2.5 Геометрична модель визначення куту

При будь-якому значенні x_s і y_s з перерахованого діапазону: $SY = NA - y_s$ і $SX = ND - x_s$. Знайдені значення відповідають довжині і висоті повернутого зображення для яких необхідно визначити відстань до контуру автомобілю. У разі, якщо знайдені значення задовольняють систему показників датчиків, координати точки S є шуканим значенням положення зображення.

2.4. Приклади результатів розрахунків та обчислення похибки

Для перевірки правильності розрахунків і визначення похибки була проведена серія тестових вимірювань показників з датчиків і реальних значень положення автомобілю при його різних розташувань відповідно оточуючого простору.

Для перевірки правильності розрахунку куту було вироблено три заміри. Оскільки розміри використовуваного приміщення не дозволяли розмістити автомобіль під великим кутом, усі величини, що перевірялися, не перевищують 20 градусів.

Таблиця 2.2

BB`	CC`	Обчислене	Дійсне	$\Delta X = \lvert X_д - X \rvert$	$\delta = \frac{\Delta X}{X_д} * 100\%$
351,71	351,41	0,2887	0,61	0,3213	52,67%
423,17	438,58	5,8225	7,43	1,6075	21,64%
399,37	371,65	15,6816	18,42	2,7384	14,87%

Середнє:	1,5557	29,72%

З отриманих значень бачимо, що абсолютна похибка для найбільш наближених до реально отриманих на практиці значень, не перевищує 2 - 3 градуси, що не буде впливати на ефективність відображення результату та для розроблюваної системи вважається прийнятним.

Для перевірки правильності розташування контуру автомобілю на схематичному зображенні використовувалися раніше перевірені значення куту повороту, разом з показниками сенсорів у той самий момент часу. Виміряні на практиці значення відповідали відстані від правого дальнього кута приміщення до правого верхнього куту прямокутнику, що оточує контур машини.

Таблиця 2.3

DX	AY	Обчислене		Дійсне		$\Delta X = \lvert X_д - X \rvert$		$\delta = \frac{\Delta X}{X_д} * 100\%$	
69,25	470,86	2,1	460,7	2,9	463,7	0,80	3,00	27,59%	0,65%
61,75	428,17	13,7	420,9	14,7	424,2	1,00	3,30	6,80%	0,78%
51,17	241,65	36,3	240,2	36,8	245,1	0,50	4,90	1,36%	2,00%
					Середнє:	0,7667	3,7333	11,92 %	1,14 %

Перевірені значення показують прийнятний рівень похибки для визначення положення автомобілю. При розміщенні фінального зображення знайдена абсолютна похибка буде практично не помітна у масштабі, який використано для відображення положення.

Крім розрахованих похибок при обчисленнях також слід враховувати похибки самих вимірювань. Величина такої похибки безпосередньо залежить від типу приладів і для поточних даних приблизно дорівнює 20 - 40 мм, згідно технічної характеристики датчиків, які було використано.

ВИСНОВКИ ДО РОЗДІЛУ 2

Розроблений алгоритм використовує доступні дані для виконання розрахунків: статичні значення розміру приміщення, відомі габарити автомобілю, показання датчиків в поточний момент часу. Для компенсації неточностей при фізичному розміщенні датчиків система створена з урахуванням динамічної зміни параметрів, за налаштування яких, відповідає користувач. Точність вимірювань і розрахунків підвищиться, при близькому до рекомендованого розміщенню всіх датчиків.

Система в режимі реального часу отримує інформацію від усіх датчиків, проводить розрахунки і формує результат, який відправляється клієнту за його запитом. На описаний процесу витрачається певна кількість часу, що викликає затримку при відображенні картинки на стороні користувача. Однак, беручи до уваги звичайну швидкість пересування автомобілю в подібних ситуаціях, затримка практично не вплине на зручність використання і працездатність системи.

Поточний алгоритм використовує в роботі показники мінімальної кількості датчиків, у розмірі чотирьох. Виходячи з економії на компонентах системи, очікувано з’являється похибка в обчисленнях. Однак, алгоритм може бути вдосконалений, при наявності додаткових даних. Збільшення кількості датчиків не тільки позитивно вплине на точність, але і підвищить швидкість роботи системи.

РОЗДІЛ 3
РОЗРОБКА СИСТЕМИ

3.1. Розробка фізичної частини та її економічна складова

Для повноцінного функціювання системи необхідна робота її фізичної частини, яка являє собою набір датчиків і сервер обробки, з яким взаємодіє кожен сенсор, а так само клієнт, який використовує систему на практиці. Оскільки вибрані для розроблюваної системи компоненти використовуються на всіх етапах її роботи, необхідно забезпечити наявність усіх складових, згідно математичного алгоритму і подальшої програмної обробки.

3.1.1. Налаштування фізичної системи

Розроблюєма система містить центральну точку збору та обробки інформації, представлену мікрокомп'ютером «Raspberry Pi», який виконує наступні функції:

- Точка доступу Wi-Fi мережі, організація мережі з унікальним ім'ям мережі та паролем доступу, до якої будуть підмикатися усі клієнти;
- Організація роботи серверу отримання і обробки даних, які будуть отримані з датчиків;
- Організація роботи front-end складової, яка формує графічний інтерфейс з отриманих даних.

Чотири датчика вимірювання відстані, об'єднаних з модулями ESP-01, функція яких у циклічному режимі вимірювати відстань до об'єкту і використовуючи модулі Wi-Fi передавати дані на сервер обробки засобом підключення до єдиної мережі.

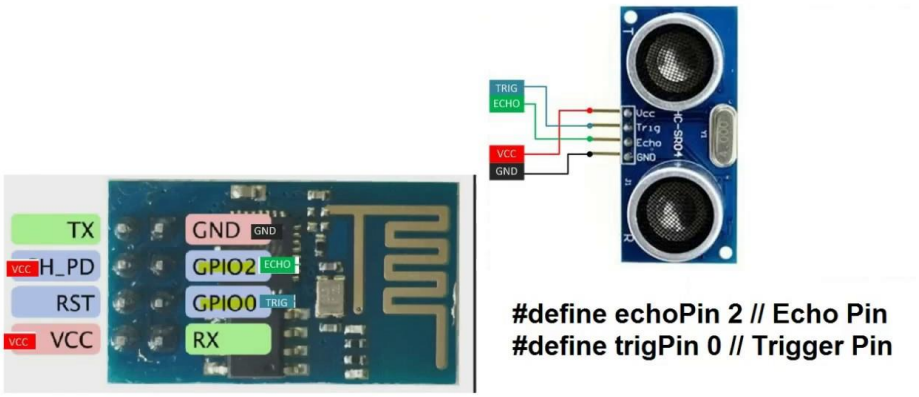


Рис. 3.1 Приклад фізичного з’єднання HC-SR04 та ESP-01

Так само складовою частиною даної системи є будь-який пристрій, який використовується клієнтом (смартфон, планшет, бортовий комп’ютер автомобіля, комп’ютер), який буде підмикатися до огранізованої мікрокомп’ютером Wi-Fi мережі і візуалізувати отриману інформацію на екрані пристрою у вигляді веб-сторінок.

3.1.2. Собівартість системи та її компонентів

Для отримання усіх необхідних для роботи даних, використовуються чотири ультразвокових сенсора відстані. У поточній версії системи обрана модель HC-SR04, середньо-ринкова ціна якої становить 0,5$ за один екземпляр. Для передачі вимірюваних даних по бездротовому з’єднанню використовуються Wi-Fi модулі для кожного ультразвукового датчику. Використана модель ESP-01 відповідає вимогам системи і реалізує передачу даних. Кожна одиниця модуля має середню вартість 50 центів і живиться від літій-іонного акумулятору, вартістю не більше 1$.

Обраний сервер обробки - мікрокомп’ютер останньої моделі «Raspberry Pi 4» має середню ціну в 40$. При цьому, з метою економії можна використати попередні версії мікрокомп’ютера, що дозволить скоротити витрати на 5$ - 10$, проте зворотньопропорційно вплине на продуктивність. Оскільки обрана версія «Raspberry» підтримує бездротове з’єднання з сенсорами, необхідність в додаткових витратах на дротове з’єднання відсутня.

Моделі «Raspberry» не містять вбудованих жорстких дисків і підтримують роботу з флеш-пам’яттю у вигляді Micro SD з довільним об’ємом. Для потреб моделі, що розробляється, достатній обсяг становить 4GB. Середня вартість подібної карти пам’яті становить 4$.

Крім основних компонент є необхідність в додаткових складових, які виступають в ролі удосконалення системи і не є критично важливими для її роботи. У число подібних надбудов входить корпус для зберігання і кріплення мікрокомп’ютера «Raspberry». Всі моделі лінійки «Raspberry Pi» йдуть в комплекті без корпусу і являють собою тільки плату. Для збереження конструкції та її безпечному розміщенні в будь-якому доступному місці була знайдена 3D модель корпусу для придбаної версії «Raspberry Pi 4».

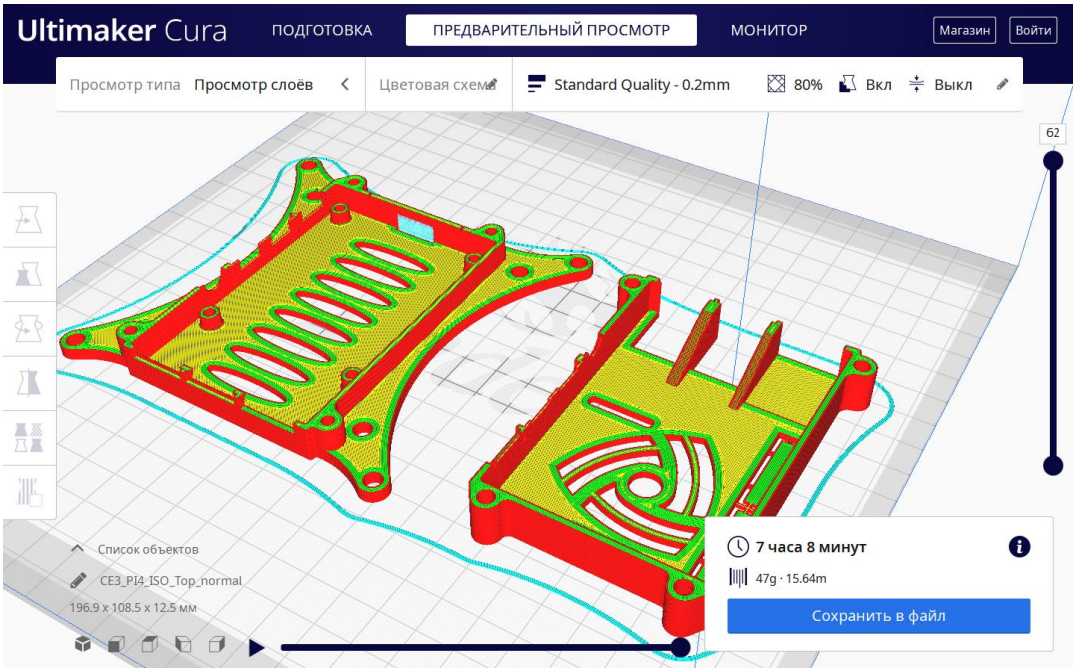


Рис. 3.2 3D модель корпусу з розрахунками витрат ресурсів на її друк

Така модель дозволяє розмістити всередині корпусу всю плату мікрокомп’ютера з безперешкодним доступом до роз’ємів. Плата надійно закріплюється на підставі корпусу і закривається кришкою, що запобігає можливості механічних пошкоджень. Так само, підстава містить додаткову секцію, що дозволяє прикручувати всю конструкцію в будь-яке необхідне місце, у тому числі до стіни.

На ринку існують аналоги корпусу для моделі «Raspberry Pi», однак ціна подібного товару, яка становить у середньому 10$, сильно перевищує собівартість самостійного друку (за наявності такої можливості). Таким чином, для друку вказаної моделі було використано філаментного пластику, оціночною вартістю приблизно в 0,5$, з додавання витрат на електроенергію для семигодинної роботи від блоку живлення в 24W. Сумарно собівартість друкованого продукту не перевищує 1$.

З мінімальною собівартістю, так само варто врахувати використання витратного матеріалу, наприклад: болтів для закріплення мікрокомп'ютера до корпусу і стіни, додаткові дроти для подачі живлення на датчики і мікрокомп'ютер. Сумарна вартість додаткових матеріалів не перевищує 0,1$.

Об'єднуючи витрати на необхідні компоненти в загальну таблицю, маємо приблизну собівартість всієї системи:

Таблиця 3.1

Компонент	Кількість	Собівартість за одиницю
Датчик HC-SR04	4	0,5 $
Wi-Fi модуль ESP-01	4	0,5 $
Акумулятор LiIo 3,7V	4	1,0 $
Raspberry Pi 4	1	40,0 $
Micro SD	1	4,0 $
Самодельный корпус	1	1,0 $
Расходники	1	0,1 $

Загальна	47,1 $

Найближчий аналог розроблюваної системи – парктроник з зоною сканування в передній і задній частині машини і цифровим відображенням даних має середню ринкову ціну в 100-150 $. Така ціна в 2 - 3 рази перевищує собівартість розробленої системи. Хоча, розміщення парктроника на машині є більш універсальним засобом для водія при паркуванні автомобіля в будь-якому місці, у випадках використання декількох автомобілів, подібна

система вимагає установки на кожен з них, що множить вартість на кількість автомобілів.

Розрахунки показують, що при усіх обмеженнях при використанні, розроблена система має собівартість нижче найближчих аналогів, що дає можливість в її удосконаленні, використовуючи більш якісне обладнання. Тим самим, збільшуючи її точність та продуктивність.

3.2. Розробка серверної частини

3.2.1. Загальна система

Архітектура серверної складової містить кілька рівнів роботи з даними.

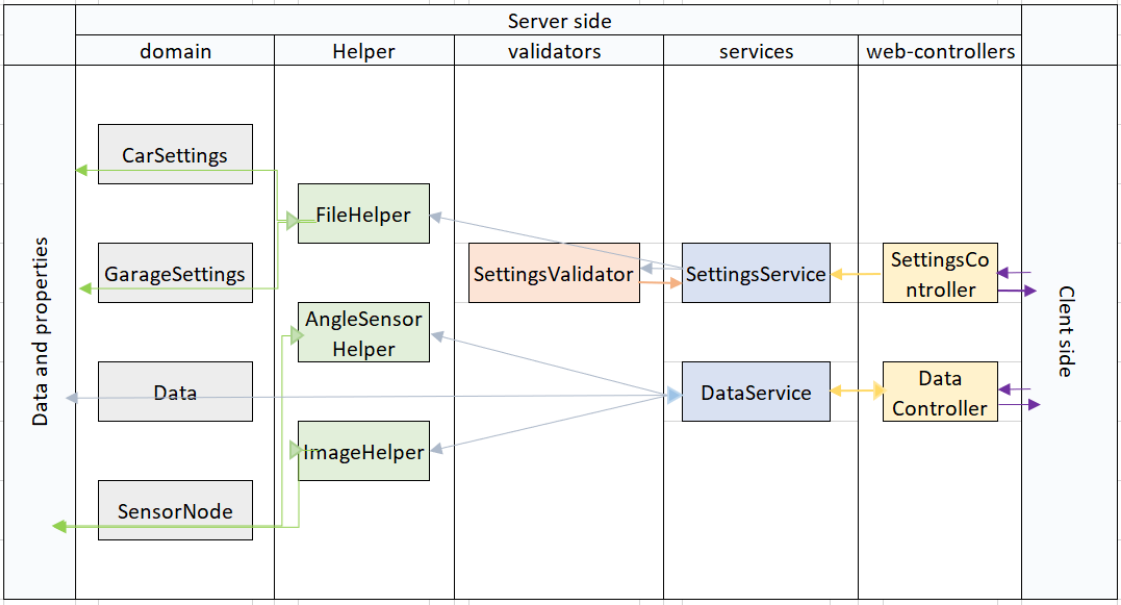


Рис. 3.3 Модель архітектури серверної складової

Методи рівня web-controller отримують параметри запиту і передають управління запитом до класів service-package, які при необхідності перевіряють її на коректність, використовуючи відповідні класи з validator-package. Далі виконується обробка даних, використовуючи класи helper-package та отримані результати готуються для відправки відповіді. Класи helper-package містять утілітні методи, які виконують допоміжну логіку в обробці даних. Усі перераховані вище рівні оперують у своїй роботі з сутностями з domain-package, які представляють собою класи зі структурою даних для повернення на сторону клієнта.

3.2.2. Структура веб-контролерів (controller package)

За допомогою фреймворке «Spring» для обробки відповідних запитів, що надходять на сервер до конкретної кінцевій точці (англ. endpoint), створюється controller-клас з методами, які відповідають одержаним запитам. Кожен метод має унікальну сигнатуру і анотацію, що визначає точку відправлення запиту. Так, при запиті на зміну даних на стороні серверу використовується анотація @PatchMapping, а для отримання інформації від сервера – @GetMapping. За логікою використання та для зручності читання і підтримки коду методи згруповані в два контролер класи.

Перший контролер – DataController з базовим шляхом звернення «/api/data» приймає запити від клієнту, пов’язані з отриманням оброблених та необроблених даних з сенсорів.

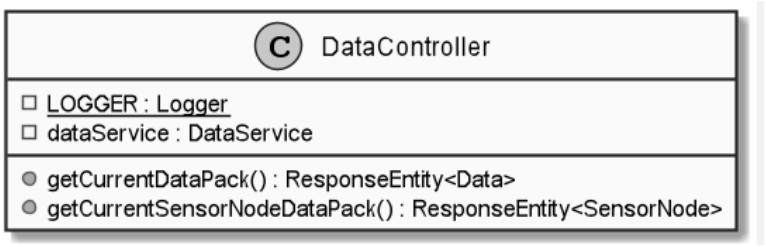


Рис. 3.4 Діаграма класу DataController

Контролер містить два методи:

- getCurrentDataPack – метод, що обробляє запит від клієнту на отримання поточних параметрів відображення автомобілю щодо оточуючого приміщення. Метод не має додаткового шляху звернення, і використовує базовий шлях класу при get-запитах;
- getCurrentSensorNodeDataPack – другий метод, що виконується при отриманні запиту на відправку поточних показників датчиків без математичної обробки. Використовується при необхідності відображення клієнту самих значень показників. Метод має додаткову приставку до шляху звернення, яка в сукупності з get-запитом формує кінцеву точку «/api/data/sensor-node».

Другий контролер SettingsController виконує дії, пов’язані з підготовкою середовища до використання. Так, контролер через проміжні рівні має доступ до даних налаштувань, збережених у файлах конфігурації. Залежно від запиту з боку клієнту дані можуть бути змінені або ж повернуті у поточному вигляді на відображення користувачеві.

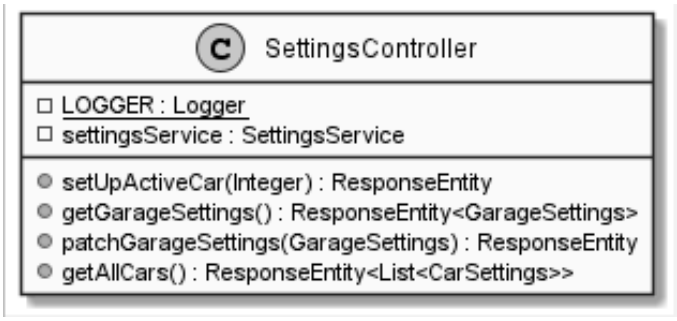


Рис. 3.5 Діаграма класу SettingsController

Контролер містить чотири методи для обробки запитів з базовим шляхом звернення «/api/settings»:

- setUpActiveCar – метод, що обробляє post-запит від клієнту з переданим числовим параметром, який позначає унікальний ідентифікатор збережених налаштувань автомобіля у файлі конфігурації. Метод встановлює дані налаштувань для використання при математичній обробці даних сенсорів. Метод використовує шлях звернення класу разом з post-запитом. При успішному застосуванні конфігурацій, клієнтові у якості відповіді повертається лише ідентифікатор статусу успішної операції, без додаткових даних;
- getGarageSettings – метод, який за get-запитом клієнту разом зі зверненням до базового шляху класу повертає поточні параметри конфігурації вимірювань приміщення і розташування сенсорів;
- getAllCars – метод, який дозволяє відправити клієнту усі наявні у файлі конфігурації параметри автомобілів. Метод доступний при get-запиті з приставкою «/cars» до базового шляху класу контролеру;

- patchGarageSettings – метод, що дозволяє клієнту змінювати конфігурацію приміщення і положення сенсорів в ньому. Оскільки метод безпосередньо редагує дані файлу конфігурацій, для нього використаний patch-запит з базовим зверненням класу. При цьому до методу у якості параметрів передається об’єкт налаштувань, отриманий від клієнта і перетворений в клас, який використовується для запису даних до файлу.

3.2.3. Структура класів сервесів (service package)

Кожен з контролерів має сервіс, який виконує базовий набір дій для виконання запиту. Зазвичай, такий набір включає в себе валідацію вхідних даних (якщо такі присутні), звернення до класів хелперів і формування об’єкту відповіді. При цьому, валідація даних відбувається за допомогою звернення до відповідних класів валідаторів. Основна логіка обробки даних розташована у класах хелперах, а сервіс лише викликає обробники в необхідному порядку для отримання кінцевого результату. У випадках, коли необхідності у класі хелпері, для виконання мінімальної частини обробки, немає – логіка розташовується безпосередньо в методі сервісу.

Перший сервіс SettingsService, об’єкт якого створюється і використовується в контролері SettingsController:

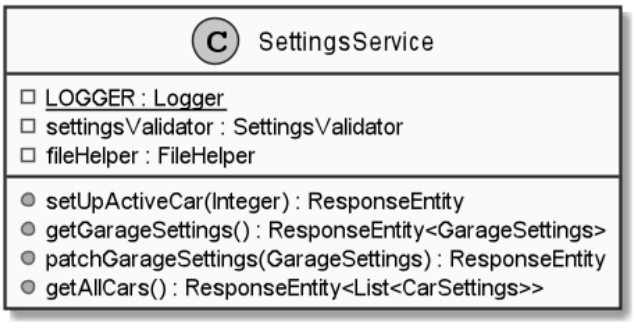


Рис. 3.6 Діаграма класу SettingsService

У класі реалізовано чотири однойменних методи, що відповідають запитам, які приймаються у контролері. Клас також містить об’єкт валідатору settingsValidator і хелперу fileHelper, методи яких використовуються для реалізації обробки запитів.

- setUpActiveCar – метод призначений для ініціалізації параметрів розрахунку значеннями, які відповідають переданим клієнтом ідентифікатора автомобіля. В першу чергу отриманий ідентифікатор передається до валідатору, і тільки в разі успішного закінчення методу перевірки відбувається безпосереднє зчитування зазначених налаштувань з файлу конфігурації. Отримані налаштування записуються в змінні, які використовуються при виконанні математичної логіки. І тільки при безпомилковому виконанні запису, метод закінчує свою роботу, передаючи управління назад до контролера, який повертає сигнал про успішне застосування налаштувань;
- getGarageSettings – метод викликає необхідну логіку формування об’єкту налаштувань з відповідного хелпер-класу. Після отримання результату, повертає дані контролеру;
- patchGarageSettings – в першу чергу метод вілідує дані, які до нього надходять, за допомогою відповідного валідатору. Якщо помилок не було виявлено і метод валідатору завершив своє виконання викликається setGarageSettings з хелпер-класу для виконання логіки запису нових параметрів;
- getAllCars – формує список з усіх наявних у файлі конфігурації параметрів автомобілів, використовуючи метод хелпер-класу. Перед поверненням результату в контролер перетворює рядкові значення полів до верхнього регістру для відповідного відображення їх на стороні клієнту. З огляду на інкрементну індексацію ідентифікаторів автомобілів, використовується метод з хелпера для кожного значення індексу.

Другий сервіс-клас DataService використовується відповідним Data-контролером для роботи із запитами, пов’язаними з даними сенсорів і математичної їх обробки. Для виклику методів математичної логіки сервіс використовує об’єкти класів хелперів: AngleSensorHelper і ImageHelper.

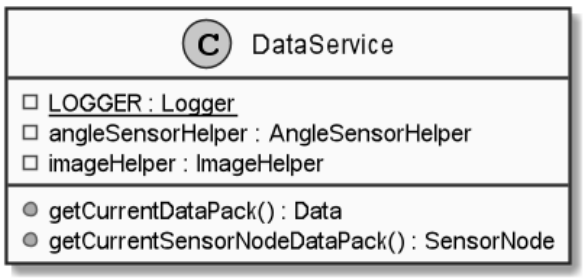


Рис. 3.7 Діаграма класу DataService

Два методи відповідають запитам, які приймають контролер класи і розпоряджаються основною логікою їх виконання:

- getCurrentDataPack – викликаючи необхідні методи з хелпер-класів, відбувається формування результуючого об’єкта з даними, який при успішному створенні буде повернутий до контролеру;
- getCurrentSensorNodeDataPack – метод, який звертається до масиву даних сенсорів і повертає останній набір даних без жодних модифікацій.

3.2.4. Структура класу валідатору (validator package)

Клас валідатор у разі необхідності першим отримує на обробку дані від клієнту з сервіс-класу. Суть роботи методів полягає в перевірці призначених для користувача даних на правильність і зупинці обробки запиту, у разі будь-якої помилки. Таким чином, класи отримують дані на обробку відразу ж, після початку їх обробки на стороні серверу. Рання перевірка забезпечує стабільність роботи системи і виключає можливість потрапляння невірних даних на подальшу обробку і безпосередньо в файл конфігурацій.

Оскільки в поточній реалізації системи єдиними вхідними даними, що вводяться клієнтом, є змінювані параметри приміщення, реалізовано єдиний клас валідатор – SettingsValidator.

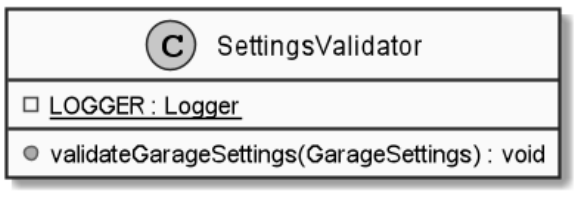


Рис. 3.8 Діаграма класу SettingsValidator

Його єдиний метод викликається з сервісу, який займається обробкою запит на зміну параметрів, перед передачею даних на подальшу обробку системою. Метод має назву validateGarageSettings і приймає як параметр сутність типу GarageSettings. Сам метод виконує перевірки всіх полів переданого параметра згідно логічним вимогам до значень. У разі невиконання однієї з умов метод повертає помилку, яка і буде повернута клієнтові у якості відповіді на запит. Обробка самого запиту сервісом буде перервана.

3.2.5. Структура класів хелперів (helper package)

Основне призначення класів хелперів – виконувати математичну і логічну частину алгоритму роботи з даними. Всі публічні методи класів між собою не пов’язані, а являють незалежний набір дій, який при виконанні буде містити логічно-завершений результат. Методи класів можуть бути використані з будь-якої точки системи і незалежно від цього результат залишиться незмінним. Приватні методи належать класу, і можуть бути викликані тільки з інших методів цього класу, містять додаткову логіку, необхідну публічним методам, але формують лише проміжний результат.

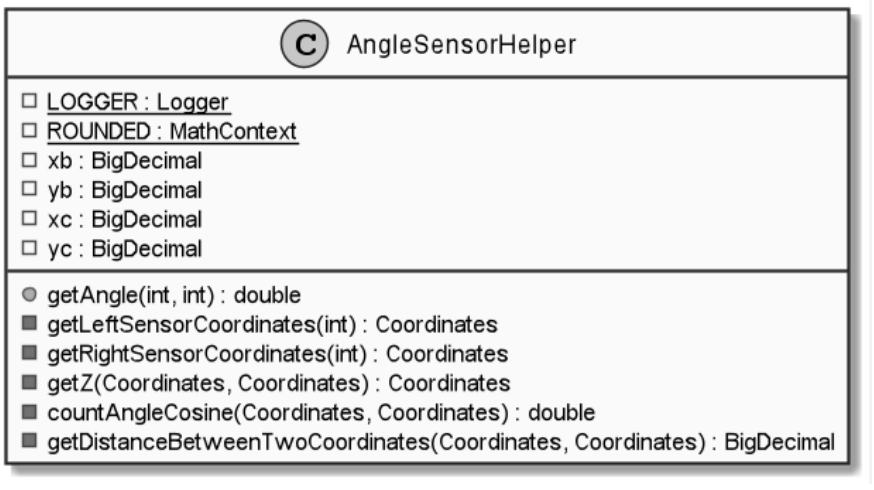


Рис. 3.9 Діаграма класу AngleSensorHelper

Клас AngleSensorHelper має єдиний публічний метод getAngle, який реалізує математичну логіку визначення куту повороту автомобіля щодо бічної частини приміщення. Метод приймає на вхід два значення показників кутових сенсорів і використовуючи встановлені налаштування розмірів

приміщення виконує розрахунки. Перераховані на діаграмі приватні методи відносяться до логіки розрахунку кута, і винесені в окремі методи з метою спрощення читання коду і розбиття алгоритму на логічні дії. Окремо кожен із приватних методів не виконує логічного блоку.

Клас ImageHelper відповідає за визначення параметрів зображення, яке з'являється на стороні клієнту. Маючи дані сенсорів і результати математичних обчислень, клас виробляє дії за розрахунками ключових характеристик зображення, що використовується для зазначеного автомобіля.

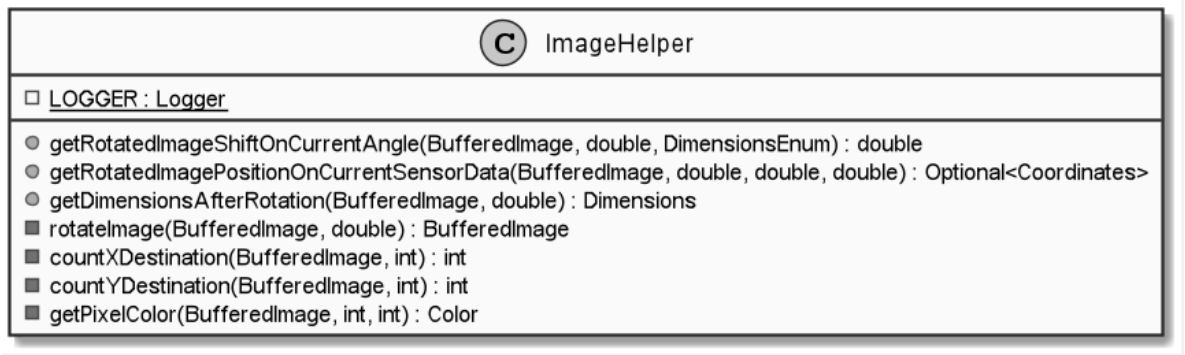


Рис. 3.10 Діаграма класу ImageHelper

Визначено три ключових методи, які додатково використовують приватні методи класу, в які винесена додаткова логіка.

- getRotatedImageShiftOnCurrentAngle – розраховує відступи від верхньої і лівої межи приміщення, щоб компенсувати поворот зображення і встановити вже трансформоване зображення на початкову позицію щодо верхнього лівого кута рамки;
- getRotatedImagePositionOnCurrentSensorData – маючи результати обробки даних сенсорів визначає початкову позицію відтворення зображення автомобіля щодо межи приміщення;
- getDimensionsAfterRotation – визначає нові розміри рамки зображення, при повороті на обчислений кут. На вхід до методу передається зображення без повороту і значення кута в радіанах.

3.3. Структура класів сутностей

Класи сутності представляють собою java-класиб які містять тільки поля, зазвичай оголошені з рівнем доступу private та методи get і set для кожного з полів. Методів, які реалізують логічну частину алгоритму в таких класах не оголошено, проте реалізовані стандартні методи класів разом з конструктором класу:

- Конструктор класу – метод, що викликається при ініціалізації нового об’єкта. Може не приймати параметрів, при цьому ініціалізувавши усі поля класу порожніми або значеннями за замовчуванням. Якщо передаються параметри, встановлює значення полів відповідно до отриманих даних;
- hashCode – вираховує значення відповідне числовому відображенню усіх значень, що містяться в полях об’єкта класу;
- equals – визначає логіку порівняння об’єктів одного класу;
- toString – об’єднує усі поля класу в рядок, для зручності виведення на консоль.

Чотири класи сутності використовуються в якості моделей, що відправляються як результат роботи сервера у відповідь клієнту. Між собою сутності не взаємопов’язані і використовуються для об’єднання примітивних значень в логічні блоки відповідей.

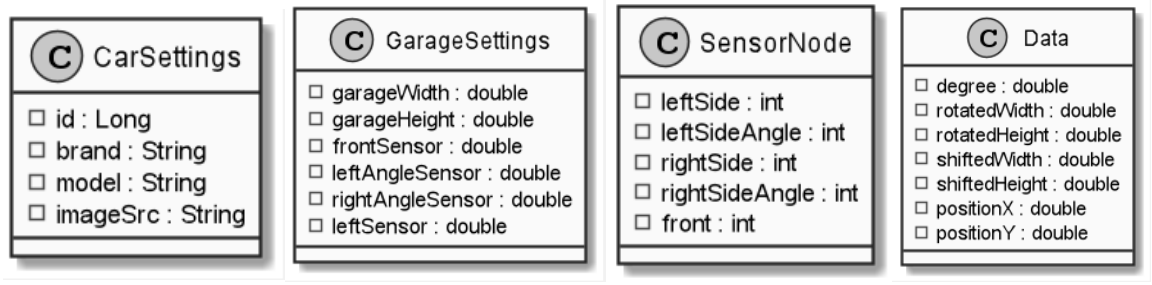


Рис. 3.11 Діаграма класів сутностей

- CarSettings – модель для зберігання об’єктів автомобілів з конфігураційного файлу. Зберігає ідентифікатор автомобіля,

текстову інформацію про бренд і марку, а також шлях до файлу з графічним зображенням;

- GarageSettings – модель для зберігання значень параметрів приміщення з конфігураційного файлу. Зберігає розміри ширини і довжини приміщення, а також положення чотирьох сенсорів щодо верхнього лівого кута;
- SensorNode – модель, яка об'єднує показники всіх сенсорів з однаковою тимчасовою міткою в один об'єкт;
- Data – модель, що зберігає результати обчислень усіх необхідних значень для відображення результату на стороні клієнта.

3.4. Розробка сторони клієнту

3.4.1. Загальна структура

Модуль клієнту приймає від користувача дані у вигляді дій на сторінці або формах введення, які налаштовані на отримання даних, очікуваних на стороні сервера. Загалом, структура веб-сайту складається з двох основних сторінок з динамічно оновлюваною інформацією.

Головна сторінка є обкладинкою сайту і надає основну інформацію про призначення веб-додатку, разом з посиланнями на сторінку «Garage». Друга сторінка складається з трьох вкладинок, які надають безпосередній функціонал роботи з системою.

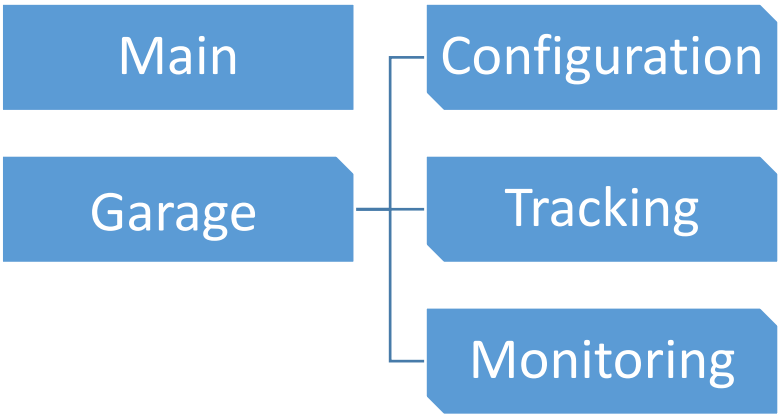


Рис. 3.12 Ієрархія сторінок

На вкладинці «Configuration» (укр. конфігурація) користувач має можливість ввести бажані налаштування приміщення і розміщення сенсорів у ньому; вкладинка «Tracking» (укр. спостереження) демонструє графіку просування обраного типу автомобіля в просторі; вкладка «Monitoring» (укр. моніторинг) надає інформацію про стан автомобіля в цифровому вигляді, відображаючи дані кожного з працюючих датчиків.

Загальна структура кожної сторінки складається з чотирьох розділів, відповідних HTML-тегами: nav, header, main і footer, кожна з яких визначена в своїй js компоненті. Розділ nav визначений в Navigation компоненті, містить головне меню сайту, при цьому для мобільної версії передбачено вертикальне розміщення пунктів, а для широкоформатних моніторів – горизонтальне. Розділ header містить ключову інформацію для кожної зі сторінок, так для головної сторінки – це текстове наповнення, а для сторінки з функціоналом – функціональні кнопки для переміщення між вкладинками. Розділ main містить основне наповнення сторінки з динамічним стилем, для зручності використання на мобільних пристроях з вертикальним форматом екрану. Розділи main і header унікальні для кожної сторінки, тому визначені в компонентах для кожної з них. Компонента Footer містить однойменний розділ, завжди розташована під основною частиною сторінки і включає допоміжну інформацію: коопірайт, посилання і рік випуску.

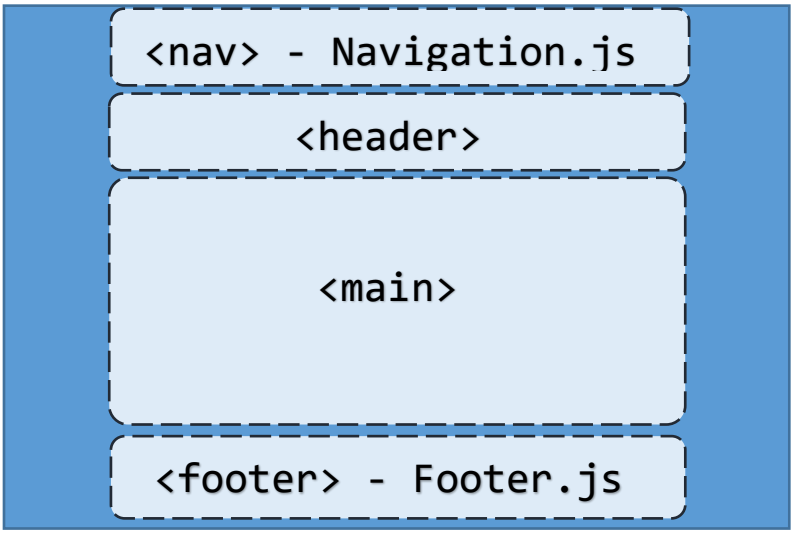


Рис. 3.13 Структурна схема компонентів сторінки

Після відправки даних на сервер, клієнт очікує пакету з результатом обробки запиту. Дані, отримані від серверу можна розділити в ієрархії, яку і використовують сторінки клієнт-модуля для відображення поточного стану системи.

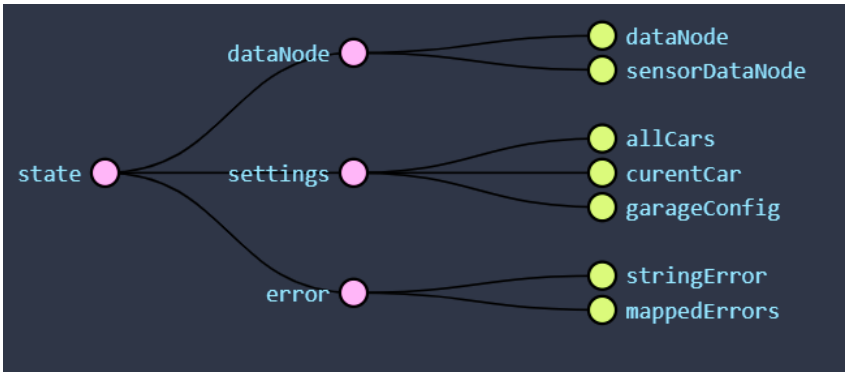


Рис. 3.14 Ієрархія даних у стані системи

Після отримання HTTP-відповіді, необхідні дані записуються у відповідні поля стану системи, при цьому поля без даних залишаються порожніми.

- state.dataNode – об’єкт стану, в якому зберігаються дані, пов’язані з роботою сенсорів. DataNode містить інформацію про обчислюваних значеннях, що залежать від показників, а в sensorDataNode зберігаються отриманні значення;
- state.settings – об’єкт стану, який зберігає значення з конфігураційного файлу. Параметр allCars містить масив з даними про доступні в системі автомобілі, currentCar - об’єкт з інформацією про автомобіль, що використовується параметрами у поточний момент часу. GarageConfig - містить поточні настройки габаритів приміщення і точок розміщення датчиків;
- state.error – об’єкт з двома полями, що відносяться до строковому типу даних опису помилки і набору «поле» - «причина» при поверненні помилки з боку сервера.

3.5. Структура головної сторінки

Структура головної сторінки визначена в компоненті Landing.js і має кореневий шлях доступу після імені хосту. При відтворенні компоненти на сервер відправляється запит на отримання інформації про усі доступні автівок з конфігураційного файлу. Після отримання відповіді від сервера дані записуються у відповідне поле state.settings.allCars стану системи, звідки і зчитуються компонентою.

Рис. 3.15 Приклад об’єкту відповіді на запит конфігурацій усіх автівок

Після успішного завантаження масиву автомобілів, для кожного з них створюється свій об’єкт дочірньої компоненти CarSlide.js, в яку передаються необхідні для відображення параметри для представлення блоків з зображенням, текстом та посиланням. Посилання має значення формату «{hostname}/garage{currentCarId}» і веде на іншу сторінку сайту з встановленою конфігурацією автівки, яка відповідає зазначеному ідентифікатору.

Кожен з блоків компоненти CarSlide додається до слайдеру, для зручної навігації по сторінці. Слайдер є HTML-блоком, який не виходить за межі екрану, але відображає увесь вміст за допомогою горизонтального, циклічного скролінгу. Для широких екранів передбачено одночасне розміщення чотирьох блоків CarSlide, і при поступовому зменшенні ширини екрану значення зміниться до одного блоку. Якщо усі завантажені блоки не розміщуються в поточному значенні ширини екрану, активується

автоматичний скролінг раз у три секунди, у якості привернення уваги та інформування користувача про наявність додаткових опцій.

3.6. Структура сторінки «Garage»

Друга сторінка визначена в компоненті Garage.js і має динамічне посилання типу «{hostname}/garage{currentCarId}». На посиланні такого типу currentCarId є змінною, за допомогою якої компоненті передається значення, необхідне для запиту відповідних даних з серверу. При завантаженні такої компоненти значення currentCarId визначається із посилання і передається у якості параметру запиту серверу на установку необхідних параметрів автівки, яка буде використовуватися у поточному сеанс. У відповідь на запит в поле state.settings.currentCar записуються дані поточного автомобіля або ж помилка (state.error.stringError), у тому разі, якщо вказаний ідентифікатор не був знайдений в конфігураціях. При успішному застосуванні налаштувань автомобіля, відправляється запит на отримання даних конфігурації приміщення для його відтворення на сторінці. Отримані параметри завантажуються в поле state.settings.garageSettings стану системи.

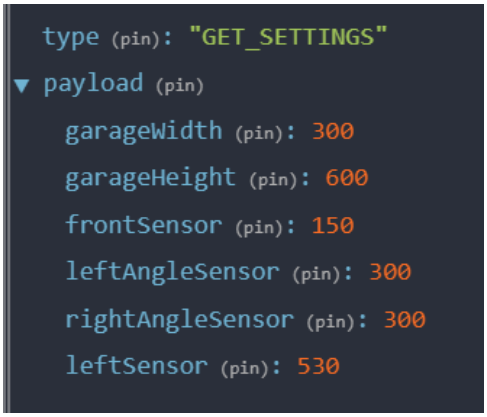


Рис. 3.16 Приклад об’єкту відповіді на запит конфігурацій приміщення

За замовчуванням, для поточної сторінки відкривається вкладинка «Спостереження», для відображення якої два рази в секунду на сервер надсилається запит про отримання нових даних сенсорів. Значення з отриманого об’єкта застосовуються в якості налаштувань css стилю HTML-

тегів з ідентифікаторами Car і Before. Тег Car є обгорткою до основного об’єкта Before із зображенням автомобіля.

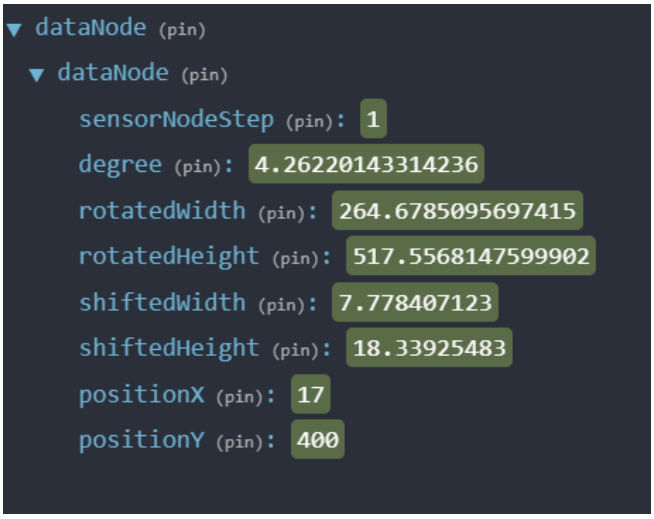


Рис. 3.17 Приклад об’єкту відповіді на запит оброблених даних

Зіставляючи отримані параметри характеристикам відображення об’єктів, формуються динамічні css стилі після кожного із запитів на сторону сервера. Таким чином створюється графічна ілюстрація руху автомобіля в режимі реального часу.

Таблиця 3.2

Тег id	css	state
#car	width	dataNode.rotatedWidth + "px"
	height	dataNode.rotatedHeight + "px"
	top	dataNode.positionY + "px"
	left	dataNode.positionX + "px"
#before	background-image	"url(" + settings.currentCar.url+")"
	transform	"rotate(" + dataNode.degree + "deg)"
	top	dataNode.shiftedWidth + "px"
	left	dataNode.shiftedHeight + "px"

На вкладинці «Monitoring» (укр. моніторинг) з частотою, аналогічною до вкладки «Tracking» (укр. спостереження), що дорівнює двом запитам в секунду, до сервера відправляється запит на отримання даних сенсорів без

математичної обробки. Отримана інформація завантажується в поле стану state.dataNode.sensorDataNode, звідки згодом відображається на сторінці в установленому місці. Залежно від значення показників застосовується різний колір фарбування блоку відповідного сенсора.

Рис. 3.18 Діаграма залежності кольору від значення

При перемиканні на третю вкладинку, для її відтворення використовуються вже завантажені в поле стану дані state.settings.garageSettings. Використовуючи ці значення відбувається пропорційне відтворення положення сенсорів і розмірів периметра, а так само встановлюються початкові дані в поля форми.

При змінах даних в будь-якому з полів відбувається тільки повторне відтворення графічної схеми, однак запит на зміну налаштувань сервера буде відправлений тільки при натисканні на кнопку підтвердження форми. Друга кнопка, одночасно з відправкою запиту, встановлює в усі поля значення замовчуванням.

3.7. Загальна структура взаємодії компонентів

Відповідно до використаних систем було розроблено два модуля і система сенсорів, які взаємодіють між собою. Сторона клієнта приймає дані від користувача за допомогою форм введення даних, формує пакет з HTTP-запитом і відправляє його на сервер. Серверна сторона приймає запит, проводить валідацію отриманих параметрів і даних з тіла запиту. У разі некоректного запиту на сторону клієнта повертається HTTP-відповідь з помилкою. У разі коректного запиту останні дані, що надійшли від сенсорів, беруться на обробку сервером. Надалі, значення разом з даними конфігурації використовуються для виконання математичних обчислень, і формування результату, очікуваного до отримання на стороні клієнта. Після успішного

завершення всіх розрахунків, відправляється HTTP-відповідь. Сторона клієнта розпаковує отриманий пакет від сервера і відображає інформацію, яку запитує користувач, в презентабельному вигляді, або ж, у разі помилки, повідомляє користувачеві про можливі проблеми в запиті.

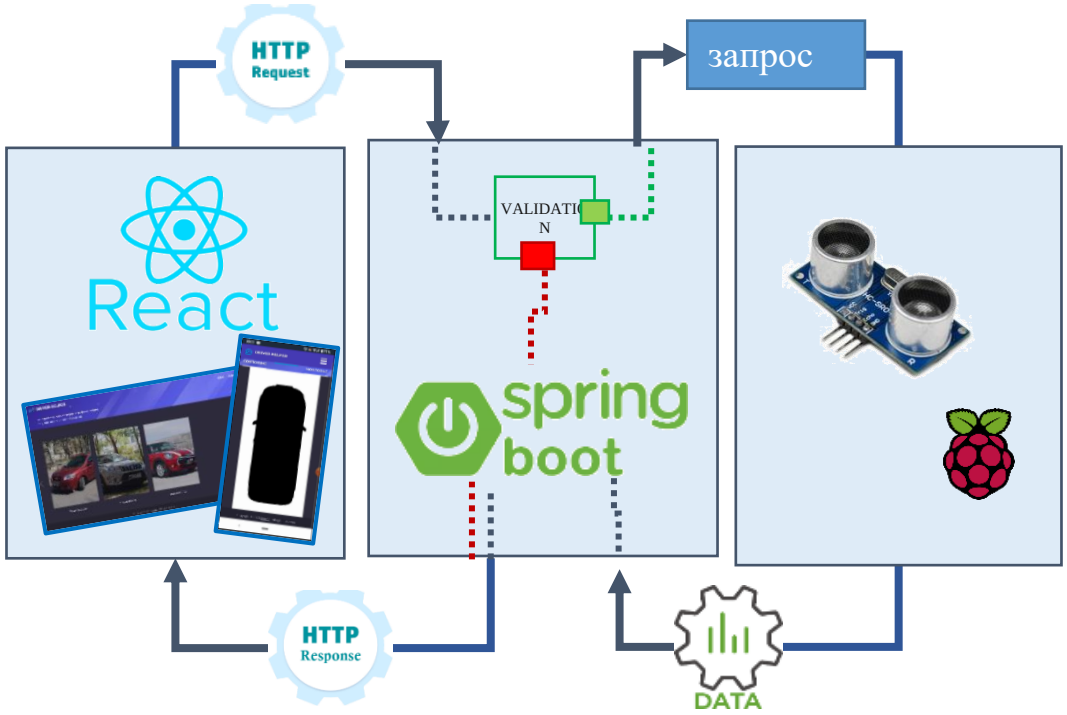


Рис. 3.19 Ілюстрація схеми взаємодії компонентів

ВИСНОВКИ ДО РОЗДІЛУ 3

Оскільки в якості програмного забезпечення, яке розробляється для системи, обрана реалізація за допомогою веб-сторінок, було реалізовано клієнт-серверну архітектуру програми. Розроблені модулі серверної і клієнтських частин взаємодіють між собою за допомогою HTTP протоколу, відправляючи пакети з запитами і відповідями.

Для реалізації серверної частини програми використаний Spring Framework, який має вбудовану MVC-платформу для веб-додатків. Існуючий в ній діспатчер сервлет виконує роль головного контролера і направляє виконання HTTP-запиту, отриманого сервером, в потрібний метод. Тим самим забезпечується простота обробки всіх необхідних запитів, створенням окремих методів для кожного з них.

Для розробки боку клієнта використовувався React JS Framework, який дозволяє не тільки створити повноцінні і працездатні веб-сторінки, а також спростити їх розробку. Кожна зі сторінок виконана в окремому js класі, з визначенням всіх функцій і використаних скриптів. Framework має безліч бібліотек з гарантованою робочим функціоналом, що дає можливість зосередитися на розробці необхідного функціоналу

РОЗДІЛ 4
ІНСТРУКЦІЯ КОРИСТУВАЧА

Для початку роботи з системою, необхідно відкрити робочу адресу через будь-який браузер на доступному пристрої. На головній сторінці користувач буде бачити невеликий блок тексту з описом системи і доступні варіанти автомобілів, підтримувані цією версією системи. Залежно від роздільності екрану робочого пристрою, користувач побачить на екрані від одного до чотирьох варіантів автівок, з можливістю гортання з боку в бік, для пошуку необхідного.

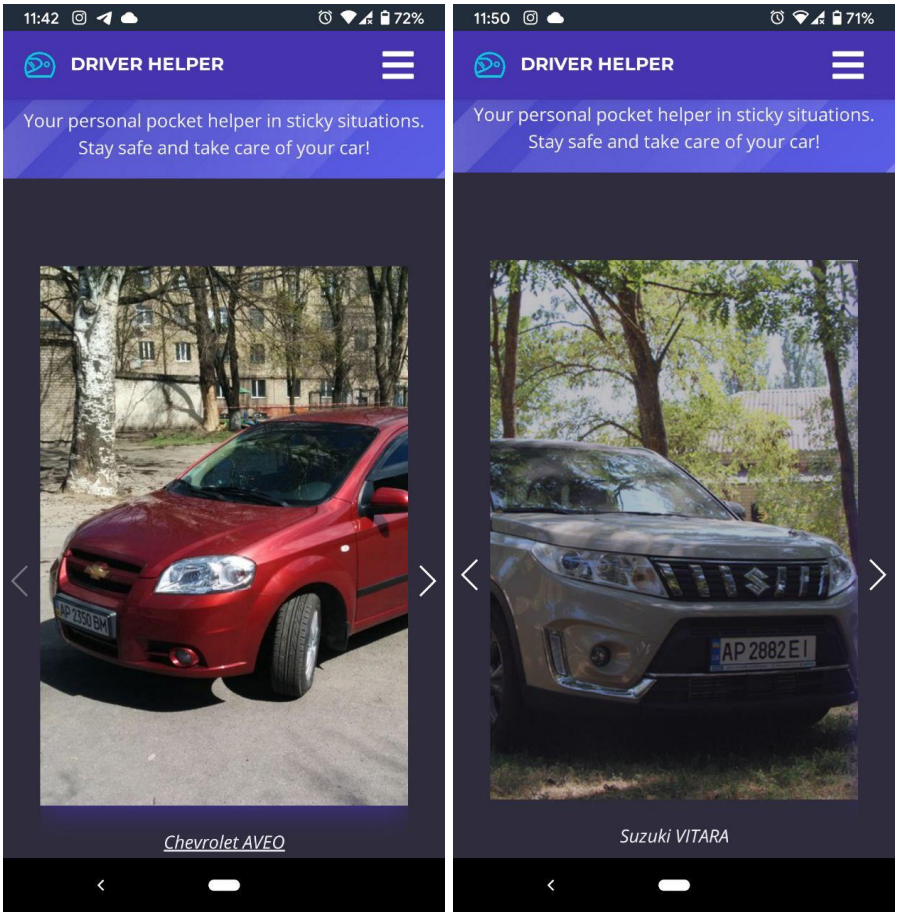


Рис. 4.1 Приклад головної сторінки сайту

При виборі першого автомобіля «Chevrolet Aveo» відкриється наступна сторінка сайту «Garage». За замовчуванням, відкривається вкладинка «Tracking» (укр. спостереження), на якій користувач бачитиме графічне відображення обраної машини щодо навколишнього приміщення, з частотою

оновлення два рази на секунду. При цьому, пропорції відтвореного приміщення відповідають поточними даними файлу конфігурації, а так само зображення автомобіля є точним контуром обраної моделі при ракурсі зверху.

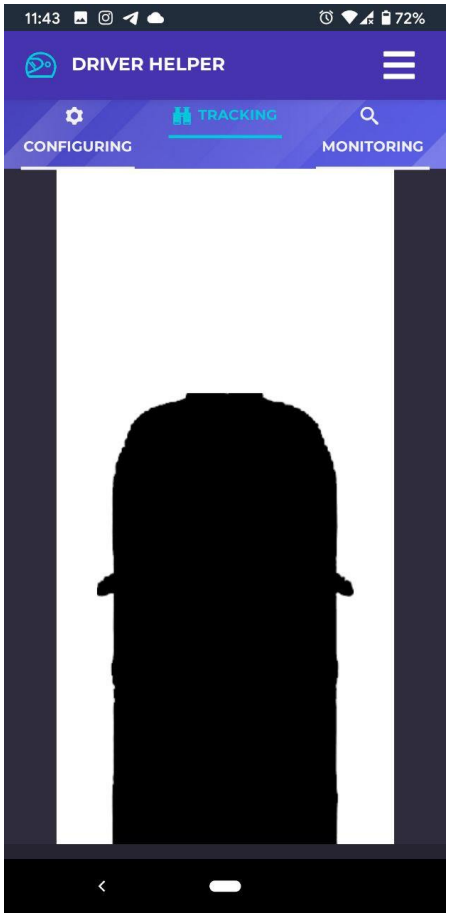


Рис. 4.2 Приклад сторінки «Garage» на вкладинці «Tracking»

Поточна сторінка, крім основного меню, містить додаткові три вкладки:

- Configuration (укр. конфігурація)
- Tracking (укр. спостереження)
- Monitoring (укр. моніторинг)

Поточна відкрита вкладинка має синій колір шрифту, коли дві доступні до переміщення – білі. Натискання на будь-яку з цих вкладинок поміняє зміст тіла сторінки, не перевантажуючи її повністю.

При переході на вкладинку «Monitoring» на екрані відобразиться периметр, аналогічний «Tracking», відповідний розмірами приміщення. Замість графічного відображення автомобіля користувачеві надані дані про

статус і показниках доступних сенсорів. Кожен сенсор зображений умовним символом кола і розташований на периметрі відповідно до вказаними налаштуванням. Для спрощення сприйняття, поруч з умовним позначенням так само присутній коротка назва і напрямок його роботи у вигляді стрілки. З метою привернення уваги, колір поряд з блоком кожного з сенсорів залежить від показників. При оптимальних значеннях шрифт має зелену заливку, при небезпечно-низьких значеннях – червону, інші випадки – жовтий колір заливки. Якщо будь-який з сенсорів з яких-небудь причин перестане передавати сигнал його блок перефарбується в сірий колір і зникне анімація навколо кола.

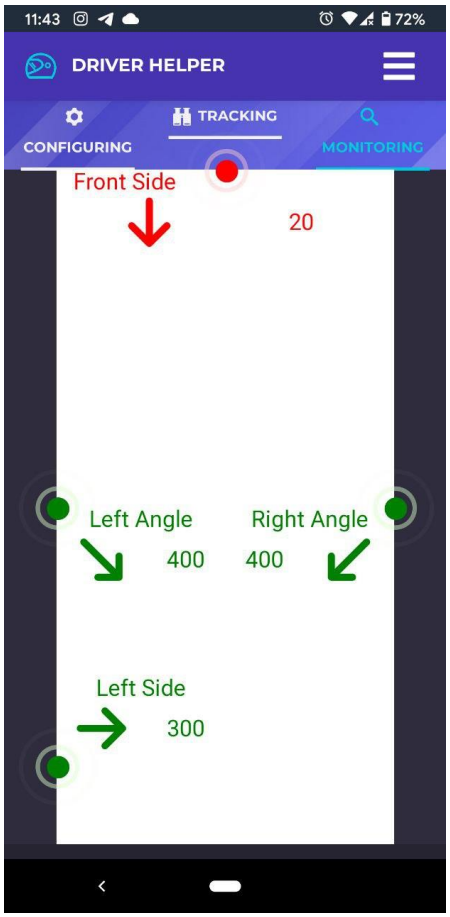


Рис. 4.3 Приклад сторінки «Garage» на вкладинці «Monitoring»

При переході на третю вкладку «Configuration» всередині периметра, аналогічного попереднім вкладинках «Tracking» та «Monitoring» відобразиться форма введення конфігурації приміщення і датчиків в ньому.

Форма складається з шести полів, відповідних кожному зі змінюваних значень.

Таблиця 4.1

Поле	Описание	
Garage Width	Довжина приміщення у ширину	300
Garage Height	Довжина приміщення у висоту	600
Front sensor position	Відстань від лівого верхнього кута приміщення до місця розміщення сенсору на задній стінці	150
Left Angle Sensor Position	Відстань від лівого верхнього кута приміщення до сенсору на лівій боковій стінці	300
Right Angle Sensor Position	Відстань від правого верхнього кута приміщення до сенсору на правій боковій стінці	300
Left Sensor Position	Відстань від лівого верхнього кута приміщення дл сенсору на лівій боковій стінці	530

При відкритті сторінки кожне поле буде містити поточне значення. Кнопка «Submit» відправить на сервер запит з введеними в поля значеннями. Кнопка «Restore Default» встановить в кожне з полів значення за замовчуванням і відправить на сервер запит з цими значеннями. Будь-який запит на зміну даних скасувати не можна, дані будуть змінені в файлі конфігурації і повернути попередню версію користувач зможе лише повторно відправивши запит з раніше використовуваними даними.

Для візуалізації параметрів периметр і умовні позначення сенсорів відображаються на сторінці в пропорційному співвідношенні до введених значень. Однак, якщо значення не входить в проміжок дозволених – для поточного поля візуальне відображення буде недоступно.

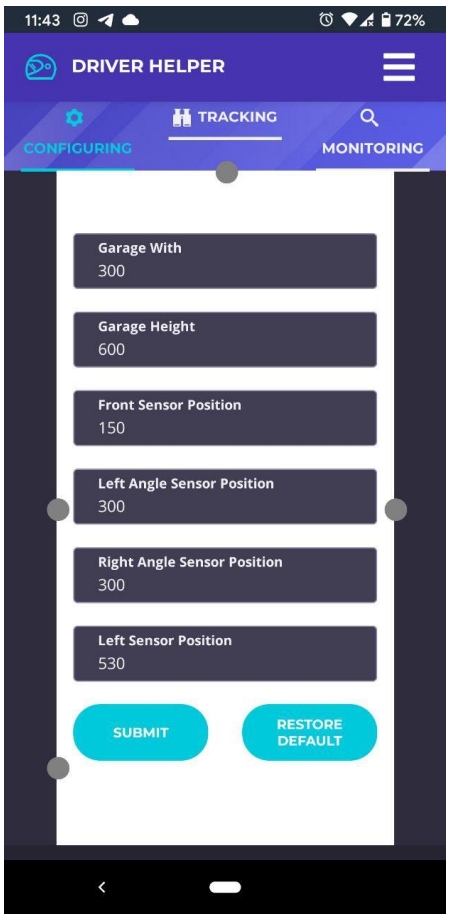


Рис. 4.4 Приклад сторінки «Garage» на вкладинці «Configuration»

Для швидкого переміщення між сторінками розроблено меню, розташоване на горі сторінок. Для відкриття меню на мобільному пристрої необхідно натиснути на іконку «меню» з трьох паралельних ліній ( ) у верхньому правому куті екрану. Спочатку меню складається з двох пунктів «Home» і «Garage», другий пункт є каскадним і після натискання розгортається другий рівень, що містить швидкі переходи між конфігураціями активного автомобіля, які відповідають вибору автомобіля на головній сторінці. Для того щоб закрити меню необхідно натиснути на іконку хрестика( ), яка з'являється на місці іконки « » при відкритті меню.

Крім мобільної версії сторінок передбачено розміщення елементів для моніторів з великою роздільністю. У таких випадках меню перетворюється з вертикального формату в горизонтальний без необхідності в додатковій кнопці для його відображення.

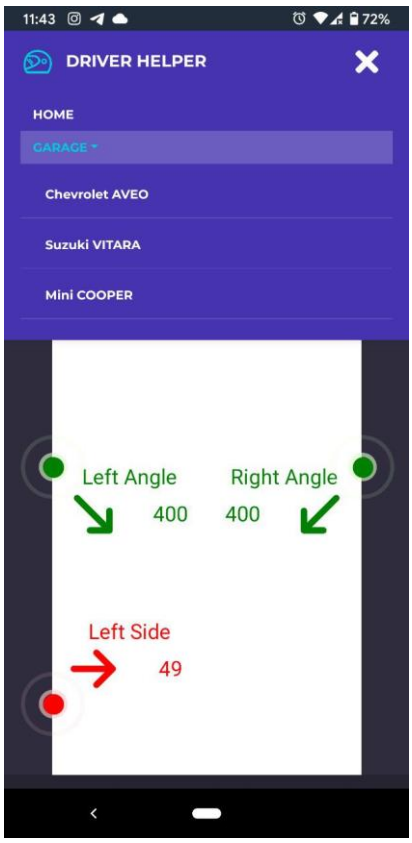


Рис. 4.5 Приклад сторінки «Garage» з відкритим мобільним меню

На вкладках зі стилем для великої роздільності інформація, яка раніше розміщувалась всередині периметру, буде відображена місці, що з’явилося за його межами (крім графічного відображення переміщення автомобілю).

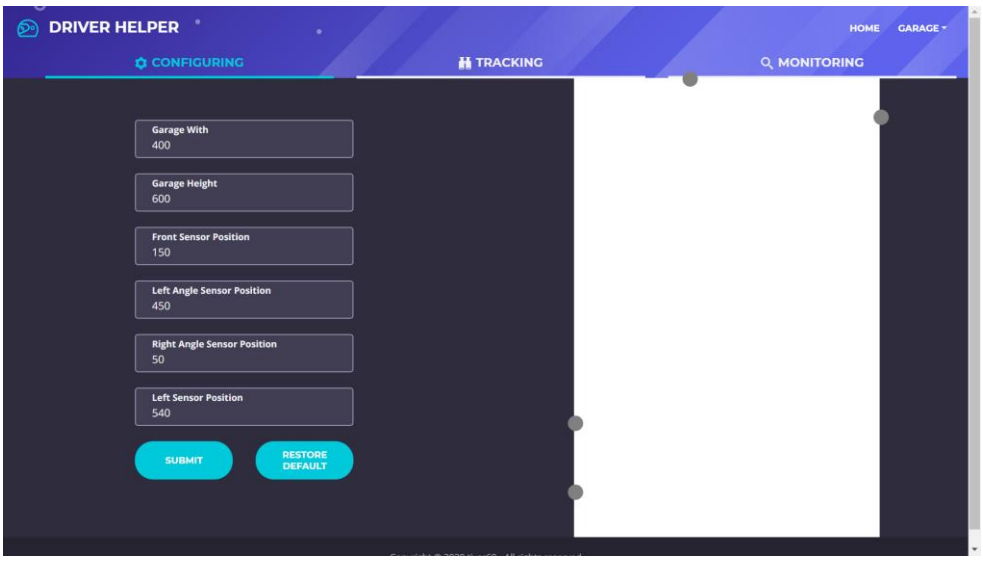


Рис. 4.6 Приклад сторінки «Garage» з для моніторів з великою роздільністю

ВИСНОВКИ ДО РОЗДІЛУ 4

У розділі було розглянуто інструкцію користувача для розробленої системи. Оскільки інтерфейс являє собою веб-сторінку, клієнт має можливість відкривати і переглядати інформацію з будь-якого доступного йому пристрою, на якому встановлено веб-браузер. З огляду на безліч пристроїв, розроблений адаптивний дизайн, що підходить під будь-які роздільності екранів.

Розглянуто можливості системи і різні сценарії роботи з нею. Користувач має доступ не тільки до інформації, що відображається, але і, використовуючи форми і керуючі компоненти, має можливість управляти і налаштовувати систему під свої потреби. Серед можливостей системи: вибір використовуваного в поточний момент автомобілю, динамічна конфігурація розмірів приміщення і положення датчиків, візуальне та цифрове відображення даних.

ВИСНОВОК

Дана робота реалізує версію системи контролю положення автомобіля в замкнутому просторі, розміри якого не набагато перевищують габарити транспортного засобу. Розроблена версія містить динамічну конфігурацію приміщення і розміщення в ньому датчиків, проте, для додавання нових автомобілів в список підтримуваних, необхідна програмна доробка. По закінченню виконання роботи можна виділити наступні висновки:

1. Поточна версія продукту реалізує повноцінний інтерфейс користувача, що дозволяє водієві в зручному для нього режимі контролювати розташування автомобіля в приміщенні. Для роботи програми в використовуваному приміщенні, згідно з розробленою схемою, було розміщено датчики і мікрокомп'ютер, на якому запущені зібрані артефакти з програмами сторони сервера і клієнту. Правильне налаштування і конфігурація мікрокомп'ютера дозволяє підключиться по бездротовому з'єднанню з будь-якого доступного пристрою, перебуваючи в безпосередній близькості.

2. Продукт був успішно протестований в реальних умовах на прикладі використання двох різних машин в одному приміщенні. І хоча, працездатність була підтверджена, додаткові тестування, з використанням іншого робочого простору і автомобілів могли б виявити дефекти в алгоритмах розрахунків і роботи з пристроями. Виправлення яких сприяє подальшому поліпшенню системи.

3. Економічна складова усіх використаних матеріалів та засобів показала виграш у вартості, відносно найближчих аналогів. Проте, отримані результати так само доводять, що система має напрямок руху для подальшого удосконалення. Використовуючи більш дорогі та ефективні прилади, які дозволять розробити новий підхід до обробки даних, система буде працювати більш стабільно, безпомилково та з більшою точністю.

Реалізація системи надала навичок роботи з фізичною системою сенсорів, їх налаштуванням та з’єднанням у єдину мережу з мікрокомп’ютером; організація роботи та програмування серверної та клієнтської частин програмного додатку системи. Створений продукт забезпечить водія надійним помічником, який дозволить не виходячи з автомобілю повністю контролювати відстані до оточуючих меж приміщення.

ЛІТЕРАТУРА

1. Ультразвук. Маленькая энциклопедия. / Глав. ред. И. П. Голямина. — М: «Советская энциклопедия», 1979. — 400 с.
2. Основы физики ультразвука, учебное пособие /В.А. Шутилов. — Изд-во Ленингр. ун-та, 1980. — 280с.
3. Многоуровневые системы клиент-сервер /Валерий Коржов. — Издательство Открытые системы, 1997. — 59с.
4. Product User's Manual – HC-SR04 Ultrasonic Sensor /Cytron Technologies Sdn. Bhd.. — 19, Jalan Kebudayaan 1A, Taman Universiti, 81300 Skudai, Johor, Malaysia: 2013. — 10с.
5. ESP-01/07/12 Series Modules User's Manual /Ai-Thinker Inc. — Shenzhen ,China: 2017. — 27с.
6. An Introduction to Client Server Computing /Subhash Chandra Yadav. — New Delhi: NEW AGE INTERNATIONAL (P) LIMITED, PUBLISHERS, 213с.
7. Микроконтроллер ESP-8266 и ультразвуковой датчик HC-SR04 подключаем к сети интернет [Електронный ресурс] /. — Електрон. текстові дан. — Режим доступу: https://mirrobo.ru/micro/wemos/
8. Raspberry Pi Documentation [Електроний ресурс] /. — Електрон. текстові дан. — Режим доступу: https://www.raspberrypi.org/documentation/
9. Spring Framework Documentation [Електроний ресурс] /. — Електрон. текстовые дан. — Режим доступу: https://docs.spring.io/spring/docs/current/spring-framework-reference/
10. React JS. Getting Started [Електроний ресурс] /. — Електрон. текстові дан. — Режим доступу: https://reactjs.org/docs/getting-started.html
11. Возраст украинского автопарка (инфографика) [Електроний ресурс] /. — Електрон. журн. — Режим доступу:

https://www.autocentre.ua/ua/avtopravo/avtobiznes/stal-izvesten-vozrast-ukrainskogo-avtoparka-infografika-41960.html

12. Что такое парктроник в автомобиле: схема и принцип работы [Електроний ресурс] /. — Електрон. текстовы дан. — Режим доступу: https://www.tts.ru/blog/tyuning/chto-takoe-parktronik-v-avtomobile-skhema-i-printsip-raboty/
13. Як правильно вибрати та купити парктронік? [Електроний ресурс] /. — Електрон. текстові дан. — Режим доступу: https://130.com.ua/uk/parktronic/

ДОДАТОК 1

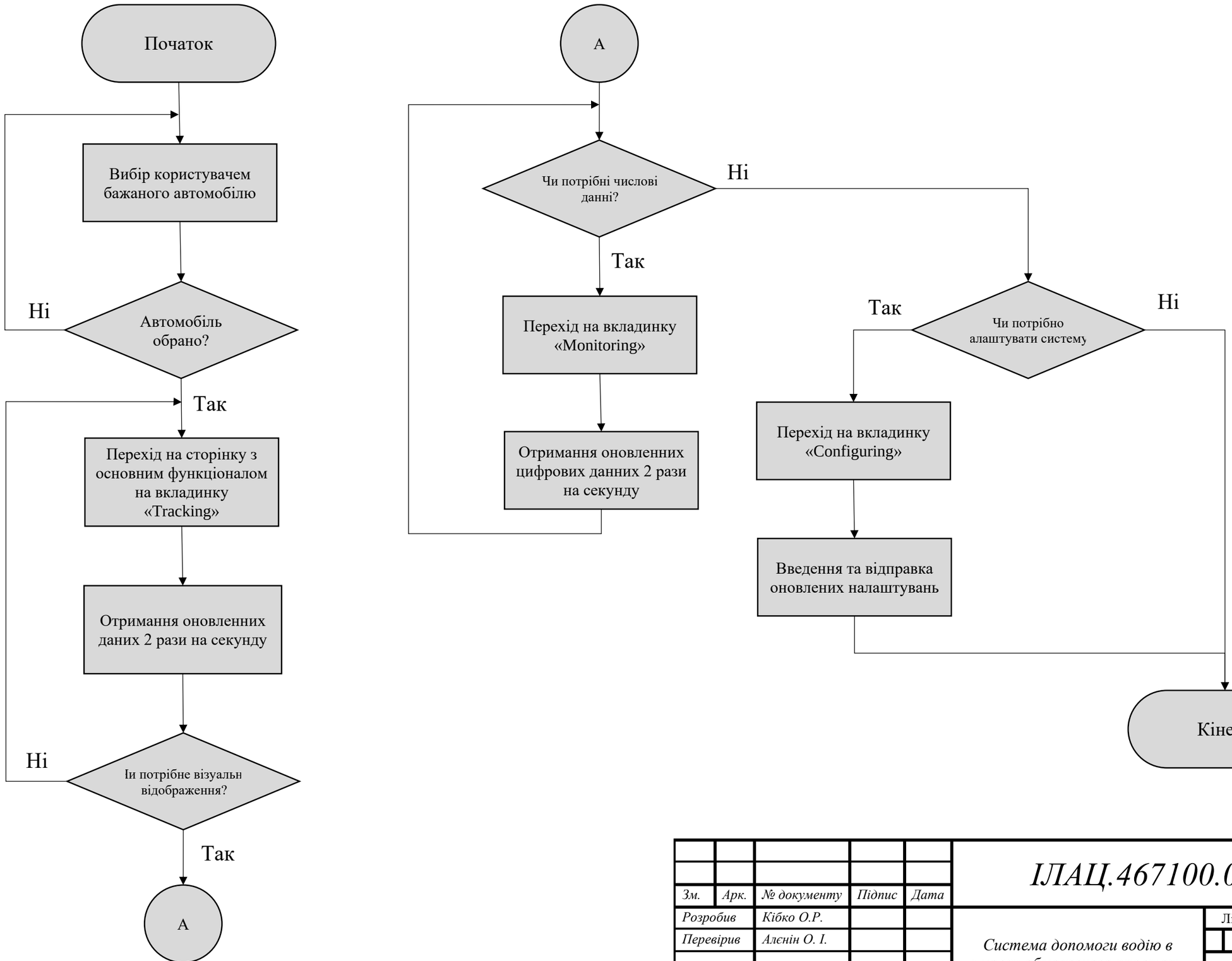
Система допомоги водію в умовах обмеженого простору

Алгоритм користувацьких дій

ІАЛЦ.467100.004 Д1

Аркушів 1

Київ 2020 р



					ІЛАЦ.467100.004 Д1		
Зм.	Арк.	№ документу	Підпис	Дата			
Розробив		Кібко О.Р.			Система допомоги водію в умовах обмеженого простору Алгоритм користувацьких дій	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Алєнін О. І.					
Н. Контр.		Симоненко В.П.				НТУУ «КПІ ім. Ігоря Сікорського», ФІОТ, гр. ІП-62	
Затверд.							

ДОДАТОК 2

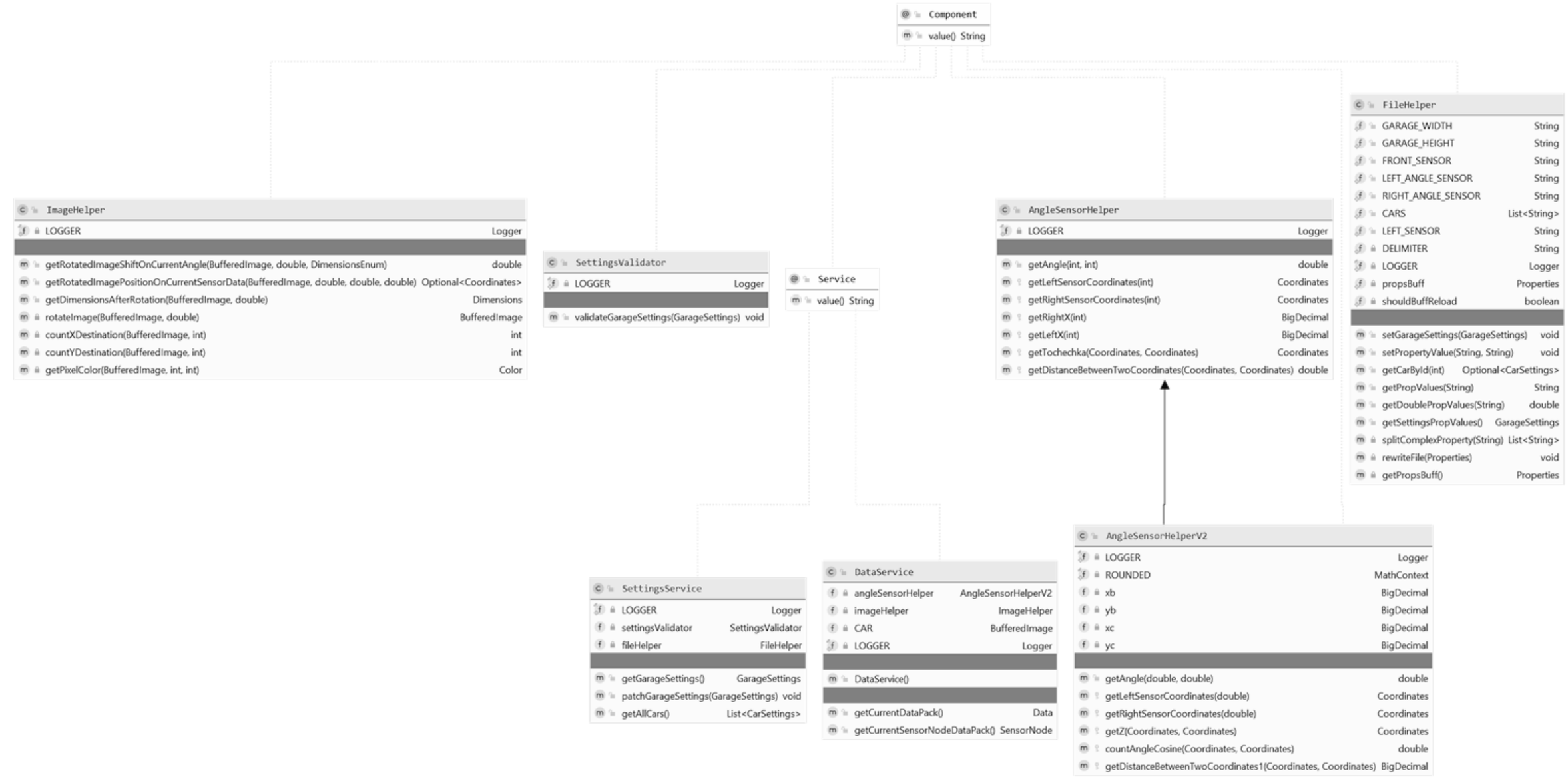
Система допомоги водію в умовах обмеженого простору

Функціональна схема

ІАЛЦ.467100.005 Д2

Аркушів 1

Київ 2020 р



					ІЛАЦ.467100.005 Д2		
Зм.	Арк.	№ документу	Підпис	Дата			
Розробив		Кібко О.Р.			Система допомоги водію в умовах обмеженого простору Функціональна схема	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Алєнін О. І.					
Н. Контр.		Симоненко В.П.				НТУУ «КПІ ім. Ігоря Сікорського», ФІОТ, гр. ІП-62	
Затверд.							

ДОДАТОК З

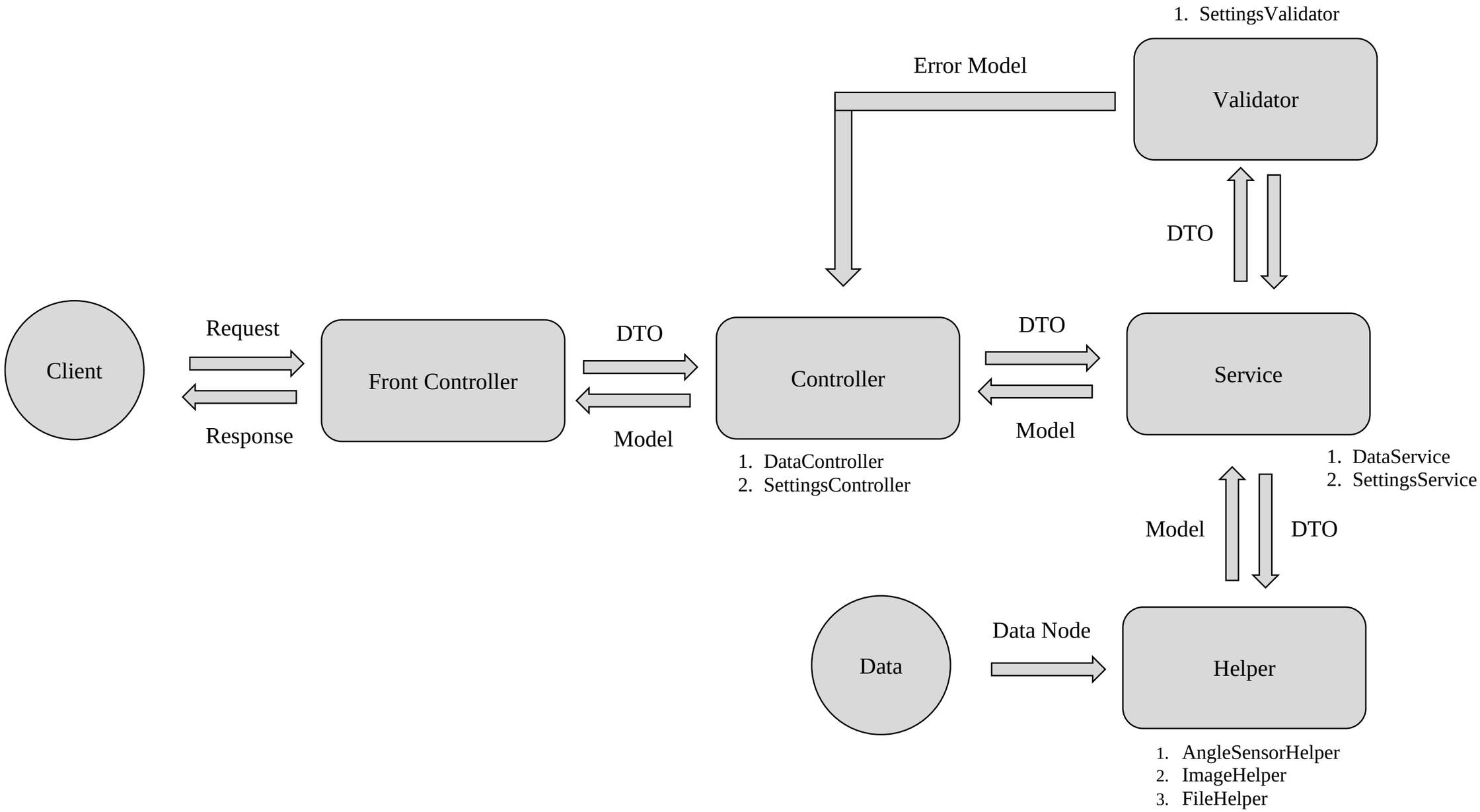
Система допомоги водію в умовах обмеженого простору

Структурна схема

ІАЛЦ.467100.006 ДЗ

Аркушів 1

Київ 2020 р



					ІЛАЦ.467100.006 ДЗ
Зм.	*Арк.*	*№ документу*	*Підпис*	*Дата*	
Розробив		*Кібко О.Р.*			*Система допомоги водію в умовах обмеженого простору Структурна схема*
Перевірив		*Алєнін О. І.*			
Н. Контр.		*Симоненко В.П.*			
Затверд.					

Літ.	Аркуш	Аркушів
	1	1

НТУУ «КПІ ім. Ігоря Сікорського», ФІОТ, гр. ІП-62

ДОДАТОК 4

Система допомоги водію в умовах обмеженого простору

Текст програми
ІАЛЦ.467100.007.Д4

Аркушів 10

Київ 2020р.

```java
package driverhelper;

@SpringBootApplication
public class DriverHelperSpringApplication {

   public static void main(String[] args) {
      SpringApplication.run(DriverHelperSpringApplication.class, args);
   }

}
```

```java
package driverhelper.controller;

@RestController
@CrossOrigin
@RequestMapping("/api/data")
public class DataController {

    private static final Logger LOGGER =
LoggerFactory.getLogger(DataController.class);

    @Autowired
    private DataService dataService;

    @GetMapping
    public ResponseEntity<?> getCurrentDataPack() {
        LOGGER.debug("Request for next data-pack");
        Data response = dataService.getCurrentDataPack();
        return new ResponseEntity<>(response, HttpStatus.OK);
    }

    @GetMapping("/sensor-node")
    public ResponseEntity<?> getCurrentSensorNodeDataPack() {
        LOGGER.debug("Request for latest sensor node data");
        SensorNode response = dataService.getCurrentSensorNodeDataPack();
        return new ResponseEntity<>(response, HttpStatus.OK);
    }

    @PostMapping("/reset") //todo: will be removed later
    public ResponseEntity<?> resetSensorDataToFirstElement() {
        LOGGER.debug("Reset test data-pack to first element");
        TestDataArray.resetSensorDataStep();
        Data response = new Data();
        response.setSensorNodeStep(TestDataArray.getCurrentSensorDataStep());
        return new ResponseEntity<>(response, HttpStatus.OK);
    }
}
```

```java
package driverhelper.controller;

@RestController
@CrossOrigin
@RequestMapping("/api/settings")
public class SettingsController {

    private static final Logger LOGGER =
LoggerFactory.getLogger(SettingsController.class);
```

```
    @Autowired
    SettingsService settingsService;

    @GetMapping //todo:replace with two methods
    public ResponseEntity<?>
setUpActiveCarAndGetGarageSettings(@RequestParam("carId") Integer carId) {
        GarageSettings response = settingsService.getGarageSettings();
        LOGGER.info("Setting car#" + carId + " as active and sending current
settings" + response);
        return new ResponseEntity<>(response, HttpStatus.OK);
    }

    @PatchMapping
    public ResponseEntity<?> patchGarageSettings(@RequestBody GarageSettings
garageSettings) {
        LOGGER.info("Receive new settings" + garageSettings.toString());
        settingsService.patchGarageSettings(garageSettings);
        return new ResponseEntity<>(HttpStatus.OK);
    }

    @GetMapping("/cars")
    public ResponseEntity<?> getAllCars() {
        LOGGER.debug("Request for all cars from data base");
        List<CarSettings> response = settingsService.getAllCars();
        return new ResponseEntity<>(response, HttpStatus.OK);
    }
}

package driverhelper.service;

@Service
public class DataService {

    @Autowired
    private AngleSensorHelperV2 angleSensorHelper;
    @Autowired
    private ImageHelper imageHelper;

    private BufferedImage CAR = ImageIO.read(new File(AVEO_IMAGE_PATH));
    private static final Logger LOGGER = LoggerFactory.getLogger(DataService.class);

    public DataService() throws IOException {
    }

    public Data getCurrentDataPack() {
        SensorNode currentSensorNode =
TestDataArray.sensorData.get(TestDataArray.getCurrentSensorDataStep());

        double currentAngle =
angleSensorHelper.getAngle(currentSensorNode.getLeftSideAngle(),
currentSensorNode.getRightSideAngle());
        Dimensions rotatedDimensions = imageHelper.getDimensionsAfterRotation(CAR,
currentAngle);
        Optional<Coordinates> rotatedPosition;
        if (currentSensorNode.getFront() > LEFT_SIDE_SENSOR_HEIGHT_POSITION ||
currentSensorNode.getLeftSide() > FRONT_SENSOR_WIDTH_POSITION) {
            rotatedPosition = Optional.of(new Coordinates());
        } else
            rotatedPosition =
imageHelper.getRotatedImagePositionOnCurrentSensorData(CAR, currentAngle,
currentSensorNode.getLeftSide(), currentSensorNode.getFront());
```

```
        TestDataArray.nextSensorDataStep();
        return Data.builder()
                .sensorNodeStep(TestDataArray.getCurrentSensorDataStep())
                .degree(Math.toDegrees(currentAngle))
                .rotatedWidth(rotatedDimensions.getWidth())
                .rotatedHeight(rotatedDimensions.getHeight())
                .positionX(rotatedPosition.get().getX())
                .positionY(rotatedPosition.get().getY())
                .shiftedWidth(imageHelper.getRotatedImageShiftOnCurrentAngle(CAR,
currentAngle, DimensionsEnum.WIDTH))
                .shiftedHeight(imageHelper.getRotatedImageShiftOnCurrentAngle(CAR,
currentAngle, DimensionsEnum.HEIGHT))
                .build();
    }

    public SensorNode getCurrentSensorNodeDataPack() {
        return
TestDataArray.sensorData.get(TestDataArray.getCurrentSensorDataStep());
    }
}
```

```
package driverhelper.service;

@Service
public class SettingsService {

    private static final Logger LOGGER =
LoggerFactory.getLogger(SettingsService.class);

    @Autowired
    private SettingsValidator settingsValidator;

    @Autowired
    private FileHelper fileHelper;

    public GarageSettings getGarageSettings() {
        GarageSettings garageSettings = fileHelper.getSettingsPropValues();
        return garageSettings;
    }

    public void patchGarageSettings(GarageSettings garageSettings) {
        settingsValidator.validateGarageSettings(garageSettings);
        fileHelper.setGarageSettings(garageSettings);
    }

    public List<CarSettings> getAllCars() {
        List<CarSettings> carList = new ArrayList<>();
        /// TODO: create method in helper
        IntStream.rangeClosed(1, 10).forEach(i -> {
            fileHelper.getCarById(i).ifPresent(carList::add);
        });
        carList.forEach(item -> item.setModel(item.getModel().toUpperCase()));
        return carList;
    }
}
```

```
package driverhelper.helper.v2;

@Component
public class AngleSensorHelperV2 extends driverhelper.helper.AngleSensorHelper {
```

```
    private static final Logger LOGGER =
LoggerFactory.getLogger(SettingsController.class);
    private static final MathContext ROUNDED = new MathContext(10);
    private BigDecimal xb = BigDecimal.valueOf(WIDTH / -2);
    private BigDecimal yb = BigDecimal.valueOf(HEIGHT -
LEFT_ANGLE_SENSOR_HEIGHT_POSITION);
    private BigDecimal xc = BigDecimal.valueOf(WIDTH / 2);
    private BigDecimal yc = BigDecimal.valueOf(HEIGHT -
RIGHT_ANGLE_SENSOR_HEIGHT_POSITION);

    /**
     * @return (360 - angle) if left sensor data is lesser than right,
     * and angle if left sensor data is bigger than right
     */
    public double getAngle(double leftAngleSensorData, double rightAngleSensorData) {
        Coordinates leftSensor = getLeftSensorCoordinates(leftAngleSensorData);
        Coordinates rightSensor = getRightSensorCoordinates(rightAngleSensorData);
        Coordinates zPoint = getZ(leftSensor, rightSensor);
        double angleCosine = countAngleCosine(rightSensor, zPoint);
        return leftSensor.getY() > rightSensor.getY() ?
                Math.toRadians(360 - Math.toDegrees(angleCosine)) :
                angleCosine;
    }

    /**
     * y = -√3x + yb + √3x_b
     */
    protected Coordinates getLeftSensorCoordinates(double distance) {
        BigDecimal resultX = xb.add(BigDecimal.valueOf(distance / 2));
        BigDecimal resultY = yb.add(BigDecimal.valueOf(Math.sqrt(3) * distance / -
2));
        return new Coordinates(resultX.doubleValue(), resultY.doubleValue());
    }

    /**
     * y = √3x + yc - √3xc
     */
    protected Coordinates getRightSensorCoordinates(double distance) {
        BigDecimal resultX = xc.subtract(BigDecimal.valueOf(distance / 2));
        BigDecimal resultY = yc.add(BigDecimal.valueOf(Math.sqrt(3) * distance / -
2));
        return new Coordinates(resultX.doubleValue(), resultY.doubleValue());
    }

    /**
     * Gets coordinates of intersection with X = xc
     * Y = (Yb-Ya)/(Xb-Xa)*X - (Yb-Ya)/(Xb-Xa)*Xa + Ya
     */
    protected Coordinates getZ(Coordinates b, Coordinates c) {
        BigDecimal kCoefficient = BigDecimal.valueOf(c.getY() -
b.getY()).divide(BigDecimal.valueOf(c.getX() - b.getX()), ROUNDED);
        BigDecimal bCoefficient = BigDecimal.valueOf(c.getX() * b.getY() - b.getX() *
c.getY()).divide(
                BigDecimal.valueOf(c.getX() - b.getX()), ROUNDED);
        return new Coordinates(xc.doubleValue(),
kCoefficient.multiply(xc).add(bCoefficient).doubleValue());
    }

    protected double countAngleCosine(Coordinates c, Coordinates z) {
        BigDecimal hypotenuse = getDistanceBetweenTwoCoordinates1(c, z);
```

```
        BigDecimal cathetus = xc.subtract(BigDecimal.valueOf(c.getX()));
        double cosine = cathetus.divide(hypotenuse, ROUNDED).doubleValue();
        return Math.acos(cosine);
    }

    /**
     * @return result^2 = (Xb-Xa)^2 + (Yb-Ya)^2
     */
    protected BigDecimal getDistanceBetweenTwoCoordinates1(Coordinates pointOne,
Coordinates pointTwo) {
        BigDecimal sum1 =
BigDecimal.valueOf(pointTwo.getX()).subtract(BigDecimal.valueOf(pointOne.getX())).pow
(2);
        BigDecimal sum2 =
BigDecimal.valueOf(pointTwo.getY()).subtract(BigDecimal.valueOf(pointOne.getY())).pow
(2);
        return sum1.add(sum2).sqrt(ROUNDED);
    }
}

package driverhelper.helper;

@Component
public class FileHelper {

    public static String GARAGE_WIDTH = "garage_width";
    public static String GARAGE_HEIGHT = "garage_height";
    public static String FRONT_SENSOR = "front_sensor";
    public static String LEFT_ANGLE_SENSOR = "left_angle_sensor";
    public static String RIGHT_ANGLE_SENSOR = "right_angle_sensor";
    public static List<String> CARS = Arrays.asList("car1", "car2", "car3", "car4",
"car5", "car6", "car7", "car8", "car9", "car10");
    public static String LEFT_SENSOR = "left_sensor";
    private static String DELIMITER = ":";
    private static final Logger LOGGER = LoggerFactory.getLogger(FileHelper.class);
    private static Properties propsBuff;
    private static boolean shouldBuffReload = false;

    public void setGarageSettings(GarageSettings garageSettings) {
        Properties props = getPropsBuff();
        props.put(GARAGE_WIDTH, String.valueOf(garageSettings.getGarageWidth()));
        props.put(GARAGE_HEIGHT, String.valueOf(garageSettings.getGarageHeight()));
        props.put(FRONT_SENSOR, String.valueOf(garageSettings.getFrontSensor()));
        props.put(LEFT_ANGLE_SENSOR,
String.valueOf(garageSettings.getLeftAngleSensor()));
        props.put(RIGHT_ANGLE_SENSOR,
String.valueOf(garageSettings.getRightAngleSensor()));
        props.put(LEFT_SENSOR, String.valueOf(garageSettings.getLeftSensor()));
        rewriteFile(props);
        LOGGER.info("Rewriting settings property file");
    }

    public void setPropertyValue(String propertyName, String value) {
        Properties props = getPropsBuff();
        props.put(propertyName, value);
        rewriteFile(props);
        LOGGER.info("Rewriting settings property file");
    }

    public String getPropValues(String propertyName) {
        Properties props = getPropsBuff();
```

```
            return props.getProperty(propertyName);
        }

        public double getDoublePropValues(String propertyName) {
            Properties props = getPropsBuff();
            return Double.parseDouble(props.getProperty(propertyName));
        }

        public GarageSettings getSettingsPropValues() {
            return GarageSettings.builder()
                    .garageWidth(getDoublePropValues(GARAGE_WIDTH))
                    .garageHeight(getDoublePropValues(GARAGE_HEIGHT))
                    .frontSensor(getDoublePropValues(FRONT_SENSOR))
                    .leftAngleSensor(getDoublePropValues(LEFT_ANGLE_SENSOR))
                    .rightAngleSensor(getDoublePropValues(RIGHT_ANGLE_SENSOR))
                    .leftSensor(getDoublePropValues(LEFT_SENSOR))
                    .build();
        }

        private List<String> splitComplexProperty(String property) {
            String[] split = property.split(DELIMITER);
            return Arrays.asList(split);
        }

        private void rewriteFile(Properties props) {
            FileOutputStream outputStream = null;
            try {
                outputStream = new FileOutputStream(SETTINGS_PROPERTIES_PATH_WRITE);
                props.store(outputStream, "Settings for driver-helper application");
            } catch (IOException e) {
                LOGGER.error("Error while writing to settings.property file");
            }
            shouldBuffReload = true;
        }

        private Properties getPropsBuff() {
            try {
                if (propsBuff == null || shouldBuffReload) {
                    propsBuff = new Properties();
                    propsBuff.load(new FileInputStream(SETTINGS_PROPERTIES_PATH_READ));
                    shouldBuffReload = false;
                }
            } catch (IOException e) {
                LOGGER.error("Error while reading settings.property file");
            }
            return propsBuff;
        }
    }

    function App() {
      return (
        <Provider store={store}>
          <Router>
            <Navigation />
            <Route exact path="/" component={Landing} />
            <Route exact path="/garage:carId" component={Garage} />
            <Footer />
          </Router>
        </Provider>
```

```
  );
}
export default App;

class Landing extends Component {
  componentDidMount() {
    this.props.getAllCars();
  }
  render() {
    const { allCars } = this.props.settings;
    var outputCarsSlides = allCars.map(car => (
      <CarSlide
        key={car.id}
        id={car.id}
        src={car.imageSrc}
        brand={car.brand}
        model={car.model}
      />
    ));
    return (
      <React.Fragment>
        <header id="header" className="header">
          <div className="header-content" style={{ paddingTop: "1.5rem" }}>
            <div className="container">
              <div className="row">
                <div className="col-lg-6">
                  <div className="text-container">
                    <p className="p-large">
                      Your personal pocket helper in sticky situations.
                      <br /> Stay safe and take care of your car!
                    </p>
                  </div>
                </div>
              </div>
            </div>
          </div>
        </header>
        <div className="slider-2">
          <div className="container">
            <div
              className="row"
              style={{
                minHeight:
                  "calc(100vh - 139.635px - 19.976px - 24px - 4.875rem)"
              }}
            >
              <div className="col-lg-12  align-self-center">
                <div className="slider-container">
                  <div className="swiper-container image-slider">
                    <div className="swiper-wrapper">{outputCarsSlides}</div>
                    <div className="swiper-button-next"></div>
                    <div className="swiper-button-prev"></div>
```

```
              </div>
            </div>
          </div>
        </div>
      </div>
    </React.Fragment>
  );
 }
}
Landing.propTypes = {
 settings: PropTypes.object.isRequired,
 getAllCars: PropTypes.func.isRequired
};
const mapStateToProps = state => ({
 settings: state.settings
});
export default connect(mapStateToProps, { getAllCars })(Landing);

class TrackingTab extends Component {
 constructor() {
  super();
  this.state = {
   i: 0,
   render: false
  };
  this.handleClick = this.handleClick.bind(this);
  this.handleResetClick = this.handleResetClick.bind(this);
 }
 handleClick(e) {
  e.preventDefault();
  this.props.getDataNode();
  console.log(this.props.dataNode);
 }
 handleResetClick(e) {
  e.preventDefault();
  this.props.postResetData();
 }
 componentDidMount() {
  this.props.postResetData();
  this.doStartResivingData();
  setInterval(this.doStartResivingData.bind(this), 1000);
 }
 async doStartResivingData() {
  if (this.state.i === 29) {
   console.log("reset");
   this.props.postResetData();
   this.setState({ i: 0 });
  } else {
   this.props.getDataNode();
   let newI = this.state.i + 1;
   console.log(newI);
```

```
      this.setState({ i: newI });
    }
  }
  render() {
    const { dataNode } = this.props.dataNode;
    const dynamicStyle0 = {
      width: dataNode.rotatedWidth + "px",
      height: dataNode.rotatedHeight + "px",
      top: dataNode.positionY + "px",
      left: dataNode.positionX + "px"
    };
    const dynamicStyle = {
      transform: "rotate(" + dataNode.degree + "deg)",
      top: dataNode.shiftedWidth + "px",
      left: dataNode.shiftedHeight + "px"
    };
    return (
      <div
        className="tab-pane fade show active"
        id="tab-2"
        role="tabpanel"
        aria-labelledby="tab-2"
      >
        <div className="container">
          <div
            className="row"
            style={{ minHeight: "calc(100vh - 94.74px - 19.976px - 24px)" }}
          >
            <div className="col-12  justify-content-md-center align-self-center">
              <div id="background">
                <div id="car" style={dynamicStyle0}>
                  <div id="before" style={dynamicStyle}></div>
                </div>
              </div>
            </div>
          </div>
        </div>
      </div>
    );
  }
}
TrackingTab.propTypes = {
  dataNode: PropTypes.object.isRequired,
  getDataNode: PropTypes.func.isRequired,
  postResetData: PropTypes.func.isRequired
};
const mapStateToProps = state => ({
  dataNode: state.dataNode
});
export default connect(mapStateToProps, { getDataNode, postResetData })(
  TrackingTab
);
```